What's new in .NET 9

Article • 02/17/2024

Learn about the new features in .NET 9 and find links to further documentation.

.NET 9, the successor to .NET 8, has a special focus on cloud-native apps and performance. It will be supported for 18 months as a standard-term support (STS) release. You can download .NET 9 here.

New for .NET 9, the engineering team posts .NET 9 preview updates on GitHub Discussions. That's a great place to ask questions and provide feedback about the release.

This article has been updated for .NET 9 Preview 1. The following sections describe the updates to the core .NET libraries in .NET 9.

Serialization

In System.Text.Json, .NET 9 has new options for serializing JSON and a new singleton that makes it easier to serialize using web defaults.

Indentation options

JsonSerializerOptions includes new properties that let you customize the indentation character and indentation size of written JSON.

C#

```
var options = new JsonSerializerOptions
{
    WriteIndented = true,
    IndentCharacter = '\t',
    IndentSize = 2,
};

string json = JsonSerializer.Serialize(
    new { Value = 1 },
    options
    );
Console.WriteLine(json);
//{
//                "Value": 1
//}
```

Default web options

If you want to serialize with the default options that ASP.NET Core uses for web apps, use the new JsonSerializerOptions.Web singleton.

C#

```
string webJson = JsonSerializer.Serialize(
    new { SomeValue = 42 },
    JsonSerializerOptions.Web // Defaults to camelCase naming policy.
    );
Console.WriteLine(webJson);
// {"someValue":42}
```

LINQ

New methods CountBy and AggregateBy have been introduced. These methods make it possible to aggregate state by key without needing to allocate intermediate groupings via GroupBy.

CountBy lets you quickly calculate the frequency of each key. The following example finds the word that occurs most frequently in a text string.

C#

```
string sourceText = """
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Sed non risus. Suspendisse lectus tortor, dignissim sit amet,
    adipiscing nec, ultricies sed, dolor. Cras elementum ultrices amet diam.
""";

// Find the most frequent word in the text.
KeyValuePair<string, int> mostFrequentWord = sourceText
    .Split(new char[] { ' ', '.', ',' },
StringSplitOptions.RemoveEmptyEntries)
    .Select(word => word.ToLowerInvariant())
    .CountBy(word => word)
    .MaxBy(pair => pair.Value);

Console.WriteLine(mostFrequentWord.Key); // amet
```

AggregateBy lets you implement more general-purpose workflows. The following example shows how you can calculate scores that are associated with a given key.

C#

```
(string id, int score)[] data =
    [
        ("0", 42),
        ("1", 5),
        ("2", 4),
        ("1", 10),
        ("0", 25),
    ];

var aggregatedData =
    data.AggregateBy(
        keySelector: entry => entry.id,
        seed: 0,
        (totalScore, curr) => totalScore + curr.score
        );

foreach (var item in aggregatedData)
{
    Console.WriteLine(item);
}
//(0, 67)
//(1, 15)
//(2, 4)
```

Index<TSource>(IEnumerable<TSource>) makes it possible to quickly extract the implicit index of an enumerable. You can now write code such as the following snippet to automatically index items in a collection.

C#

```
IEnumerable<string> lines2 = File.ReadAllLines("output.txt");
foreach ((int index, string line) in lines2.Index())
{
    Console.WriteLine($"Line number: {index + 1}, Line: {line}");
}
```

Collections

The PriorityQueue<TElement,TPriority> collection type in the System.Collections.Generic namespace includes a new Remove(TElement, TElement, TPriority, IEqualityComparer<TElement>) method that you can use the update the priority of an item in the queue.

PriorityQueue.Remove() method

.NET 6 introduced the PriorityQueue<TElement,TPriority> collection, which provides a simple and fast array-heap implementation. One issue with array heaps in general is that they don't support priority updates, which makes them prohibitive for use in algorithms such as variations of Dijkstra's algorithm.

While it's not possible to implement efficient $O(\log n)$ priority updates in the existing collection, the new PriorityQueue<TElement,TPriority>.Remove(TElement, TElement, TPriority, IEqualityComparer<TElement>) method makes it possible to emulate priority updates (albeit at $O(n)$ time):

```
C#
```

```
public static void UpdatePriority<TElement, TPriority>(
    this PriorityQueue<TElement, TPriority> queue,
    TElement element,
    TPriority priority
    )
{
    // Scan the heap for entries matching the current element.
    queue.Remove(element, out _, out _);
    // Re-insert the entry with the new priority.
    queue.Enqueue(element, priority);
}
```

This method unblocks users who want to implement graph algorithms in contexts where asymptotic performance isn't a blocker. (Such contexts include education and prototyping.) For example, here's a toy implementation of Dijkstra's algorithm that uses the new API.

Cryptography

For cryptography, .NET 9 adds a new one-shot hash method on the CryptographicOperations type. It also adds new classes that use the KMAC algorithm.

CryptographicOperations.HashData() method

.NET includes several static "one-shot" implementations of hash functions and related functions. These APIs include SHA256.HashData and HMACSHA256.HashData. One-shot APIs are preferable to use because they can provide the best possible performance and reduce or eliminate allocations.

If a developer wants to provide an API that supports hashing where the caller defines which hash algorithm to use, it's typically done by accepting a HashAlgorithmName argument. However, using that pattern with one-shot APIs would require switching over

every possible HashAlgorithmName and then using the appropriate method. To solve that problem, .NET 9 introduces the CryptographicOperations.HashData API. This API lets you produce a hash or HMAC over an input as a one-shot where the algorithm used is determined by a HashAlgorithmName.

C#

```
static void HashAndProcessData(HashAlgorithmName hashAlgorithmName, byte[]
data)
{
    byte[] hash = CryptographicOperations.HashData(hashAlgorithmName, data);
    ProcessHash(hash);
}
```

KMAC algorithm

.NET 9 provides the KMAC algorithm as specified by NIST SP-800-185. KECCAK Message Authentication Code (KMAC) is a pseudorandom function and keyed hash function based on KECCAK.

The following new classes use the KMAC algorithm. Use instances to accumulate data to produce a MAC, or use the static `HashData` method for a one-shot over a single input.

- Kmac128
- Kmac256
- KmacXof128
- KmacXof256

KMAC is available on Linux with OpenSSL 3.0 or later, and on Windows 11 Build 26016 or later. You can use the static `IsSupported` property to determine if the platform supports the desired algorithm.

C#

```
if (Kmac128.IsSupported)
{
    byte[] key = GetKmacKey();
    byte[] input = GetInputToMac();
    byte[] mac = Kmac128.HashData(key, input, outputLength: 32);
}
else
{
    // Handle scenario where KMAC isn't available.
}
```

Reflection

In .NET Core versions and .NET 5-8, support for building an assembly and emitting reflection metadata for dynamically created types was limited to a runnable AssemblyBuilder. The lack of support for *saving* an assembly was often a blocker for customers migrating from .NET Framework to .NET. .NET 9 adds public APIs to AssemblyBuilder to save an emitted assembly.

The new, persisted AssemblyBuilder implementation is runtime and platform independent. To create a persisted `AssemblyBuilder` instance, use the new AssemblyBuilder.DefinePersistedAssembly API. The existing AssemblyBuilder.DefineDynamicAssembly API accepts the assembly name and optional custom attributes. To use the new API, pass the core assembly, `System.Private.CoreLib`, which is used for referencing base runtime types. There's no option for AssemblyBuilderAccess. And for now, the persisted `AssemblyBuilder` implementation only supports saving, not running. After you create an instance of the persisted `AssemblyBuilder`, the subsequent steps for defining a module, type, method, or enum, writing IL, and all other usages remain unchanged. That means you can use existing System.Reflection.Emit code as-is for saving the assembly. The following code shows an example.

C#

```
public void CreateAndSaveAssembly(string assemblyPath)
{
    AssemblyBuilder ab = AssemblyBuilder.DefinePersistedAssembly(
        new AssemblyName("MyAssembly"),
        typeof(object).Assembly
        );
    TypeBuilder tb = ab.DefineDynamicModule("MyModule")
        .DefineType("MyType", TypeAttributes.Public | TypeAttributes.Class);

    MethodBuilder mb = tb.DefineMethod(
        "SumMethod",
        MethodAttributes.Public | MethodAttributes.Static,
        typeof(int), [typeof(int), typeof(int)]
        );
    ILGenerator il = mb.GetILGenerator();
    il.Emit(OpCodes.Ldarg_0);
    il.Emit(OpCodes.Ldarg_1);
    il.Emit(OpCodes.Add);
    il.Emit(OpCodes.Ret);

    tb.CreateType();
    ab.Save(assemblyPath); // or could save to a Stream
}
```

```
public void UseAssembly(string assemblyPath)
{
    Assembly assembly = Assembly.LoadFrom(assemblyPath);
    Type type = assembly.GetType("MyType");
    MethodInfo method = type.GetMethod("SumMethod");
    Console.WriteLine(method.Invoke(null, [5, 10]));
}
```

See also

- Our vision for .NET 9 blog post

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Breaking changes in .NET 9

Article • 02/14/2024

If you're migrating an app to .NET 9, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or Windows Forms.

This article categorizes each breaking change as *binary incompatible* or *source incompatible*, or as a *behavioral change*:

- **Binary incompatible** - When run against the new runtime or component, existing binaries may encounter a breaking change in behavior, such as failure to load or execute, and if so, require recompilation.
- **Source incompatible** - When recompiled using the new SDK or component or to target the new runtime, existing source code may require source changes to compile successfully.
- **Behavioral change** - Existing code and binaries may behave differently at run time. If the new behavior is undesirable, existing code would need to be updated and recompiled.

> **Note**
>
> This article is a work in progress. It's not a complete list of breaking changes in .NET 9. To query breaking changes that are still pending publication, see **Issues of .NET**.

Core .NET libraries

Expand table

Title	Type of change	Introduced version
Creating type of array of System.Void not allowed	Behavioral change	Preview 1
Inline array struct size limit is enforced	Behavioral change	Preview 1
InMemoryDirectoryInfo prepends rootDir to files	Behavioral change	Preview 1
RuntimeHelpers.GetSubArray returns different type	Behavioral change	Preview 1

Networking

Expand table

Title	Type of change	Introduced version
HttpListenerRequest.UserAgent is nullable	Source incompatible	Preview 1

SDK and MSBuild

Expand table

Title	Type of change	Introduced version
dotnet workload commands output change	Behavioral change	Preview 1
Terminal logger is default	Behavioral change	Preview 1

Windows Forms

Expand table

Title	Type of change	Introduced version
BindingSource.SortDescriptions doesn't return null	Behavioral change	Preview 1
Changes to nullability annotations	Source incompatible	Preview 1
ComponentDesigner.Initialize throws ArgumentNullException	Behavioral change	Preview 1
DataGridViewRowAccessibleObject.Name starting row index	Behavioral change	Preview 1
No exception if DataGridView is null	Behavioral change	Preview 1

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

What's new in .NET 8

Article • 02/08/2024

.NET 8 is the successor to .NET 7. It will be supported for three years as a long-term support (LTS) release. You can download .NET 8 here.

.NET runtime

The .NET 8 runtime includes improvements to performance, garbage collection, and the core and extension libraries. It also includes a new globalization mode for mobile apps and new source generators for COM interop and configuration binding. For more information, see What's new in the .NET 8 runtime.

.NET SDK

For information about what's new in the .NET SDK, Native AOT, code analysis, and diagnostics, see What's new in the SDK and tooling for .NET 8.

C# 12

C# 12 shipped with the .NET 8 SDK. For more information, see What's new in C# 12.

.NET Aspire

.NET Aspire is an opinionated, cloud-ready stack for building observable, production ready, distributed applications. .NET Aspire is delivered through a collection of NuGet packages that handle specific cloud-native concerns, and is available in preview for .NET 8. For more information, see .NET Aspire (Preview).

ASP.NET Core

ASP.NET Core includes improvements to Blazor, SignalR, minimal APIs, Native AOT, Kestrel and HTTP.sys servers, and authentication and authorization. For more information, see What's new in ASP.NET Core 8.0.

.NET MAUI

.NET MAUI includes new functionality for controls, gesture recognizers, Windows apps, navigation, and platform integration. It also includes some behavior changes and many performance enhancements. For more information, see What's new in .NET MAUI for .NET 8.

EF Core

Entity Framework Core includes improvements to complex type objects, collections of primitive types, JSON column mapping, raw SQL queries, lazy loading, tracked-entity access, model building, math translations, and other features. It also includes a new `HierarchyId` type. For more information, see What's New in EF Core 8.

Windows Forms

Windows Forms includes improvements to data binding, Visual Studio DPI, and high DPI. Button commands are also fully enabled now. For more information, see What's new for .NET 8 (Windows Forms).

Windows Presentation Foundation

Windows Presentation Foundation (WPF) adds the ability to use hardware acceleration and a new OpenFolderDialog control. For more information, see What's new in WPF for .NET 8.

See also

- Breaking changes in .NET 8

.NET preview announcements

- Announcing .NET 8
- Announcing .NET 8 RC 2
- Announcing .NET 8 RC 1
- Announcing .NET 8 Preview 7
- Announcing .NET 8 Preview 6
- Announcing .NET 8 Preview 5
- Announcing .NET 8 Preview 4
- Announcing .NET 8 Preview 3
- Announcing .NET 8 Preview 2

- Announcing .NET 8 Preview 1

ASP.NET Core preview announcements

- ASP.NET Core in .NET 8
- ASP.NET Core updates in .NET 8 RC 2
- ASP.NET Core updates in .NET 8 RC 1
- ASP.NET Core updates in .NET 8 Preview 7
- ASP.NET Core updates in .NET 8 Preview 6
- ASP.NET Core updates in .NET 8 Preview 5
- ASP.NET Core updates in .NET 8 Preview 4
- ASP.NET Core updates in .NET 8 Preview 3
- ASP.NET Core updates in .NET 8 Preview 2
- ASP.NET Core updates in .NET 8 Preview 1

What's new in the .NET 8 runtime

Article • 02/08/2024

This article describes new features in the .NET runtime for .NET 8.

Performance improvements

.NET 8 includes improvements to code generation and just-in time (JIT) compilation:

- Arm64 performance improvements
- SIMD improvements
- Support for AVX-512 ISA extensions (see Vector512 and AVX-512)
- Cloud-native improvements
- JIT throughput improvements
- Loop and general optimizations
- Optimized access for fields marked with ThreadStaticAttribute
- Consecutive register allocation. Arm64 has two instructions for table vector lookup, which require that all entities in their tuple operands are present in consecutive registers.
- JIT/NativeAOT can now unroll and auto-vectorize some memory operations with SIMD, such as comparison, copying, and zeroing, if it can determine their sizes at compile time.

In addition, dynamic profile-guided optimization (PGO) has been improved and is now enabled by default. You no longer need to use a runtime configuration option to enable it. Dynamic PGO works hand-in-hand with tiered compilation to further optimize code based on additional instrumentation that's put in place during tier 0.

On average, dynamic PGO increases performance by about 15%. In a benchmark suite of ~4600 tests, 23% saw performance improvements of 20% or more.

Codegen struct promotion

.NET 8 includes a new physical promotion optimization pass for codegen that generalizes the JIT's ability to promote struct variables. This optimization (also called *scalar replacement of aggregates*) replaces the fields of struct variables by primitive variables that the JIT is then able to reason about and optimize more precisely.

The JIT already supported this optimization but with several large limitations including:

- It was only supported for structs with four or fewer fields.

- It was only supported if each field was a primitive type, or a simple struct wrapping a primitive type.

Physical promotion removes these limitations, which fixes a number of long-standing JIT issues.

Garbage collection

.NET 8 adds a capability to adjust the memory limit on the fly. This is useful in cloud-service scenarios, where demand comes and goes. To be cost-effective, services should scale up and down on resource consumption as the demand fluctuates. When a service detects a decrease in demand, it can scale down resource consumption by reducing its memory limit. Previously, this would fail because the garbage collector (GC) was unaware of the change and might allocate more memory than the new limit. With this change, you can call the RefreshMemoryLimit() API to update the GC with the new memory limit.

There are some limitations to be aware of:

- On 32-bit platforms (for example, Windows x86 and Linux ARM), .NET is unable to establish a new heap hard limit if there isn't already one.
- The API might return a non-zero status code indicating the refresh failed. This can happen if the scale-down is too aggressive and leaves no room for the GC to maneuver. In this case, consider calling `GC.Collect(2, GCCollectionMode.Aggressive)` to shrink the current memory usage, and then try again.
- If you scale up the memory limit beyond the size that the GC believes the process can handle during startup, the `RefreshMemoryLimit` call will succeed, but it won't be able to use more memory than what it perceives as the limit.

The following code snippet shows how to call the API.

```
C#

GC.RefreshMemoryLimit();
```

You can also refresh some of the GC configuration settings related to the memory limit. The following code snippet sets the heap hard limit to 100 mebibytes (MiB):

```
C#

AppContext.SetData("GCHeapHardLimit", (ulong)100 * 1_024 * 1_024);
```

```
GC.RefreshMemoryLimit();
```

The API can throw an InvalidOperationException if the hard limit is invalid, for example, in the case of negative heap hard limit percentages and if the hard limit is too low. This can happen if the heap hard limit that the refresh will set, either because of new AppData settings or implied by the container memory limit changes, is lower than what's already committed.

Globalization for mobile apps

Mobile apps on iOS, tvOS, and MacCatalyst can opt into a new *hybrid* globalization mode that uses a lighter ICU bundle. In hybrid mode, globalization data is partially pulled from the ICU bundle and partially from calls into Native APIs. Hybrid mode serves all the locales supported by mobile.

Hybrid mode is most suitable for apps that can't work in invariant globalization mode and that use cultures that were trimmed from ICU data on mobile. You can also use it when you want to load a smaller ICU data file. (The *icudt_hybrid.dat* file is 34.5 % smaller than the default ICU data file *icudt.dat*.)

To use hybrid globalization mode, set the `HybridGlobalization` MSBuild property to true:

XML

```
<PropertyGroup>
  <HybridGlobalization>true</HybridGlobalization>
</PropertyGroup>
```

There are some limitations to be aware of:

- Due to limitations of Native API, not all globalization APIs are supported in hybrid mode.
- Some of the supported APIs have different behavior.

To check if your application is affected, see Behavioral differences.

Source-generated COM interop

.NET 8 includes a new source generator that supports interoperating with COM interfaces. You can use the GeneratedComInterfaceAttribute to mark an interface as a COM interface for the source generator. The source generator will then generate code

to enable calling from C# code to unmanaged code. It also generates code to enable calling from unmanaged code into C#. This source generator integrates with LibraryImportAttribute, and you can use types with the GeneratedComInterfaceAttribute as parameters and return types in `LibraryImport`-attributed methods.

C#

```
using System.Runtime.InteropServices;
using System.Runtime.InteropServices.Marshalling;

[GeneratedComInterface]
[Guid("5401c312-ab23-4dd3-aa40-3cb4b3a4683e")]
partial interface IComInterface
{
    void DoWork();
}

internal partial class MyNativeLib
{
    [LibraryImport(nameof(MyNativeLib))]
    public static partial void GetComInterface(out IComInterface
comInterface);
}
```

The source generator also supports the new GeneratedComClassAttribute attribute to enable you to pass types that implement interfaces with the GeneratedComInterfaceAttribute attribute to unmanaged code. The source generator will generate the code necessary to expose a COM object that implements the interfaces and forwards calls to the managed implementation.

Methods on interfaces with the GeneratedComInterfaceAttribute attribute support all the same types as `LibraryImportAttribute`, and `LibraryImportAttribute` now supports `GeneratedComInterface`-attributed types and `GeneratedComClass`-attributed types.

If your C# code only uses a `GeneratedComInterface`-attributed interface to either wrap a COM object from unmanaged code or wrap a managed object from C# to expose to unmanaged code, you can use the options in the Options property to customize which code will be generated. These options mean you don't need to write marshallers for scenarios that you know won't be used.

The source generator uses the new StrategyBasedComWrappers type to create and manage the COM object wrappers and the managed object wrappers. This new type handles providing the expected .NET user experience for COM interop, while providing customization points for advanced users. If your application has its own mechanism for defining types from COM or if you need to support scenarios that source-generated COM doesn't currently support, consider using the new StrategyBasedComWrappers

type to add the missing features for your scenario and get the same .NET user experience for your COM types.

If you're using Visual Studio, new analyzers and code fixes make it easy to convert your existing COM interop code to use source-generated interop. Next to each interface that has the ComImportAttribute, a lightbulb offers an option to convert to source-generated interop. The fix changes the interface to use the GeneratedComInterfaceAttribute attribute. And next to every class that implements an interface with `GeneratedComInterfaceAttribute`, a lightbulb offers an option to add the GeneratedComClassAttribute attribute to the type. Once your types are converted, you can move your `DllImport` methods to use `LibraryImportAttribute`.

Limitations

The COM source generator doesn't support apartment affinity, using the `new` keyword to activate a COM CoClass, and the following APIs:

- IDispatch-based interfaces.
- IInspectable-based interfaces.
- COM properties and events.

Configuration-binding source generator

.NET 8 introduces a source generator to provide AOT and trim-friendly configuration in ASP.NET Core. The generator is an alternative to the pre-existing reflection-based implementation.

The source generator probes for Configure(TOptions), Bind, and Get calls to retrieve type info from. When the generator is enabled in a project, the compiler implicitly chooses generated methods over the pre-existing reflection-based framework implementations.

No source code changes are needed to use the generator. It's enabled by default in AOT'd web apps. For other project types, the source generator is off by default, but you can opt in by setting the `EnableConfigurationBindingGenerator` property to `true` in your project file:

XML

```
<PropertyGroup>

<EnableConfigurationBindingGenerator>true</EnableConfigurationBindingGenerat
```

```
or>
</PropertyGroup>
```

The following code shows an example of invoking the binder.

C#

```
public class ConfigBindingSG
{
    static void RunIt(params string[] args)
    {
        WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
        IConfigurationSection section =
builder.Configuration.GetSection("MyOptions");

        // !! Configure call - to be replaced with source-gen'd
implementation
        builder.Services.Configure<MyOptions>(section);

        // !! Get call - to be replaced with source-gen'd implementation
        MyOptions? options0 = section.Get<MyOptions>();

        // !! Bind call - to be replaced with source-gen'd implementation
        MyOptions options1 = new();
        section.Bind(options1);

        WebApplication app = builder.Build();
        app.MapGet("/", () => "Hello World!");
        app.Run();
    }

    public class MyOptions
    {
        public int A { get; set; }
        public string S { get; set; }
        public byte[] Data { get; set; }
        public Dictionary<string, string> Values { get; set; }
        public List<MyClass> Values2 { get; set; }
    }

    public class MyClass
    {
        public int SomethingElse { get; set; }
    }
}
```

Core .NET libraries

This section contains the following subtopics:

- Reflection
- Serialization
- Time abstraction
- UTF8 improvements
- Methods for working with randomness
- Performance-focused types
- System.Numerics and System.Runtime.Intrinsics
- Data validation
- Metrics
- Cryptography
- Networking
- Stream-based ZipFile methods

Reflection

Function pointers were introduced in .NET 5, however, the corresponding support for reflection wasn't added at that time. When using `typeof` or reflection on a function pointer, for example, `typeof(delegate*<void>())` or `FieldInfo.FieldType` respectively, an IntPtr was returned. Starting in .NET 8, a System.Type object is returned instead. This type provides access to function pointer metadata, including the calling conventions, return type, and parameters.

> **Note**
>
> A function pointer instance, which is a physical address to a function, continues to be represented as an **IntPtr**. Only the reflection type has changed.

The new functionality is currently implemented only in the CoreCLR runtime and MetadataLoadContext.

New APIs have been added to System.Type, such as IsFunctionPointer, and to System.Reflection.PropertyInfo, System.Reflection.FieldInfo, and System.Reflection.ParameterInfo. The following code shows how to use some of the new APIs for reflection.

C#

```
using System;
using System.Reflection;

// Sample class that contains a function pointer field.
public unsafe class UClass
```

```
{
    public delegate* unmanaged[Cdecl, SuppressGCTransition]<in int, void>
_fp;
}

internal class FunctionPointerReflection
{
    public static void RunIt()
    {
        FieldInfo? fieldInfo = typeof(UClass).GetField(nameof(UClass._fp));

        // Obtain the function pointer type from a field.
        Type? fpType = fieldInfo?.FieldType;

        // New methods to determine if a type is a function pointer.
        Console.WriteLine(
        $"IsFunctionPointer: {fpType?.IsFunctionPointer}");
        Console.WriteLine(
            $"IsUnmanagedFunctionPointer:
{fpType?.IsUnmanagedFunctionPointer}");

        // New methods to obtain the return and parameter types.
        Console.WriteLine($"Return type:
{fpType?.GetFunctionPointerReturnType()}");

        if (fpType is not null)
        {
            foreach (Type parameterType in
fpType.GetFunctionPointerParameterTypes())
            {
                Console.WriteLine($"Parameter type: {parameterType}");
            }
        }

        // Access to custom modifiers and calling conventions requires a
"modified type".
        Type? modifiedType = fieldInfo?.GetModifiedFieldType();

        // A modified type forwards most members to its underlying type.
        Type? normalType = modifiedType?.UnderlyingSystemType;

        if (modifiedType is not null)
        {
            // New method to obtain the calling conventions.
            foreach (Type callConv in
modifiedType.GetFunctionPointerCallingConventions())
            {
                Console.WriteLine($"Calling convention: {callConv}");
            }
        }

        // New method to obtain the custom modifiers.
        Type[]? modifiers =
            modifiedType?.GetFunctionPointerParameterTypes()
[0].GetRequiredCustomModifiers();
```

```
        if (modifiers is not null)
        {
            foreach (Type modreq in modifiers)
            {
                Console.WriteLine($"Required modifier for first parameter:
{modreq}");
            }
        }
    }
}
```

The previous example produces the following output:

Output

```
IsFunctionPointer: True
IsUnmanagedFunctionPointer: True
Return type: System.Void
Parameter type: System.Int32&
Calling convention:
System.Runtime.CompilerServices.CallConvSuppressGCTransition
Calling convention: System.Runtime.CompilerServices.CallConvCdecl
Required modifier for first parameter:
System.Runtime.InteropServices.InAttribute
```

Serialization

Many improvements have been made to System.Text.Json serialization and deserialization functionality in .NET 8. For example, you can customize handling of members that aren't in the JSON payload.

The following sections describe other serialization improvements:

- Built-in support for additional types
- Source generator
- Interface hierarchies
- Naming policies
- Read-only properties
- Disable reflection-based default
- New JsonNode API methods
- Non-public members
- Streaming deserialization APIs
- WithAddedModifier extension method
- New JsonContent.Create overloads
- Freeze a JsonSerializerOptions instance

For more information about JSON serialization in general, see JSON serialization and deserialization in .NET.

Built-in support for additional types

The serializer has built-in support for the following additional types.

- Half, Int128, and UInt128 numeric types.

```
C#

Console.WriteLine(JsonSerializer.Serialize(
    [ Half.MaxValue, Int128.MaxValue, UInt128.MaxValue ]
));
//
[65500,170141183460469231731687303715884105727,340282366920938463463374
607431768211455]
```

- Memory<T> and ReadOnlyMemory<T> values. `byte` values are serialized to Base64 strings, and other types to JSON arrays.

```
C#

JsonSerializer.Serialize<ReadOnlyMemory<byte>>(new byte[] { 1, 2, 3 });
// "AQID"
JsonSerializer.Serialize<Memory<int>>(new int[] { 1, 2, 3 }); //
[1,2,3]
```

Source generator

.NET 8 includes enhancements of the System.Text.Json source generator that are aimed at making the Native AOT experience on par with the reflection-based serializer. For example:

- The source generator now supports serializing types with required and init properties. These were both already supported in reflection-based serialization.
- Improved formatting of source-generated code.
- JsonSourceGenerationOptionsAttribute feature parity with JsonSerializerOptions. For more information, see Specify options (source generation).
- Additional diagnostics (such as SYSLIB1034 and SYSLIB1039).
- Don't include types of ignored or inaccessible properties.

- Support for nesting `JsonSerializerContext` declarations within arbitrary type kinds.
- Support for compiler-generated or *unspeakable* types in weakly typed source generation scenarios. Since compiler-generated types can't be explicitly specified by the source generator, System.Text.Json now performs nearest-ancestor resolution at run time. This resolution determines the most appropriate supertype with which to serialize the value.
- New converter type `JsonStringEnumConverter<TEnum>`. The existing JsonStringEnumConverter class isn't supported in Native AOT. You can annotate your enum types as follows:

  C#

  ```
  [JsonConverter(typeof(JsonStringEnumConverter<MyEnum>))]
  public enum MyEnum { Value1, Value2, Value3 }

  [JsonSerializable(typeof(MyEnum))]
  public partial class MyContext : JsonSerializerContext { }
  ```

  For more information, see Serialize enum fields as strings.
- New `JsonConverter.Type` property lets you look up the type of a non-generic `JsonConverter` instance:

  C#

  ```
  Dictionary<Type, JsonConverter>
  CreateDictionary(IEnumerable<JsonConverter> converters)
      => converters.Where(converter => converter.Type != null)
                   .ToDictionary(converter => converter.Type!);
  ```

  The property is nullable since it returns `null` for `JsonConverterFactory` instances and `typeof(T)` for `JsonConverter<T>` instances.

Chain source generators

The JsonSerializerOptions class includes a new TypeInfoResolverChain property that complements the existing TypeInfoResolver property. These properties are used in contract customization for chaining source generators. The addition of the new property means that you don't have to specify all chained components at one call site—they can be added after the fact. TypeInfoResolverChain also lets you introspect the chain or remove components from it. For more information, see Combine source generators.

In addition, JsonSerializerOptions.AddContext<TContext>() is now obsolete. It's been superseded by the TypeInfoResolver and TypeInfoResolverChain properties. For more information, see SYSLIB0049.

Interface hierarchies

.NET 8 adds support for serializing properties from interface hierarchies.

The following code shows an example where the properties from both the immediately implemented interface and its base interface are serialized.

C#

```
public static void InterfaceHierarchies()
{
    IDerived value = new DerivedImplement { Base = 0, Derived = 1 };
    string json = JsonSerializer.Serialize(value);
    Console.WriteLine(json); // {"Derived":1,"Base":0}
}

public interface IBase
{
    public int Base { get; set; }
}

public interface IDerived : IBase
{
    public int Derived { get; set; }
}

public class DerivedImplement : IDerived
{
    public int Base { get; set; }
    public int Derived { get; set; }
}
```

Naming policies

JsonNamingPolicy includes new naming policies for `snake_case` (with an underscore) and `kebab-case` (with a hyphen) property name conversions. Use these policies similarly to the existing JsonNamingPolicy.CamelCase policy:

C#

```
var options = new JsonSerializerOptions
{
    PropertyNamingPolicy = JsonNamingPolicy.SnakeCaseLower
};
```

```
JsonSerializer.Serialize(new { PropertyName = "value" }, options);
// { "property_name" : "value" }
```

For more information, see Use a built-in naming policy.

Read-only properties

You can now deserialize onto read-only fields or properties (that is, those that don't have a `set` accessor).

To opt into this support globally, set a new option, PreferredObjectCreationHandling, to JsonObjectCreationHandling.Populate. If compatibility is a concern, you can also enable the functionality more granularly by placing the `[JsonObjectCreationHandling(JsonObjectCreationHandling.Populate)]` attribute on specific types whose properties are to be populated, or on individual properties.

For example, consider the following code that deserializes into a `CustomerInfo` type that has two read-only properties.

C#

```
public static void ReadOnlyProperties()
{
    CustomerInfo customer = JsonSerializer.Deserialize<CustomerInfo>("""
        { "Names":["John Doe"], "Company":{"Name":"Contoso"} }
        """)!;

    Console.WriteLine(JsonSerializer.Serialize(customer));
}

class CompanyInfo
{
    public required string Name { get; set; }
    public string? PhoneNumber { get; set; }
}

[JsonObjectCreationHandling(JsonObjectCreationHandling.Populate)]
class CustomerInfo
{
    // Both of these properties are read-only.
    public List<string> Names { get; } = new();
    public CompanyInfo Company { get; } = new()
    {
        Name = "N/A",
        PhoneNumber = "N/A"
    };
}
```

Prior to .NET 8, the input values were ignored and the `Names` and `Company` properties retained their default values.

Output

```
{"Names":[],"Company":{"Name":"N/A","PhoneNumber":"N/A"}}
```

Now, the input values are used to populate the read-only properties during deserialization.

Output

```
{"Names":["John Doe"],"Company":{"Name":"Contoso","PhoneNumber":"N/A"}}
```

For more information about the *populate* deserialization behavior, see Populate initialized properties.

Disable reflection-based default

You can now disable using the reflection-based serializer by default. This disablement is useful to avoid accidental rooting of reflection components that aren't even in use, especially in trimmed and Native AOT apps. To disable default reflection-based serialization by requiring that a JsonSerializerOptions argument be passed to the JsonSerializer serialization and deserialization methods, set the `JsonSerializerIsReflectionEnabledByDefault` MSBuild property to `false` in your project file.

Use the new IsReflectionEnabledByDefault API to check the value of the feature switch. If you're a library author building on top of System.Text.Json, you can rely on the property to configure your defaults without accidentally rooting reflection components.

For more information, see Disable reflection defaults.

New JsonNode API methods

The JsonNode and System.Text.Json.Nodes.JsonArray types include the following new methods.

C#

```
public partial class JsonNode
{
    // Creates a deep clone of the current node and all its descendants.
```

```
    public JsonNode DeepClone();

    // Returns true if the two nodes are equivalent JSON representations.
    public static bool DeepEquals(JsonNode? node1, JsonNode? node2);

    // Determines the JsonValueKind of the current node.
    public JsonValueKind GetValueKind(JsonSerializerOptions options = null);

    // If node is the value of a property in the parent
    // object, returns its name.
    // Throws InvalidOperationException otherwise.
    public string GetPropertyName();

    // If node is the element of a parent JsonArray,
    // returns its index.
    // Throws InvalidOperationException otherwise.
    public int GetElementIndex();

    // Replaces this instance with a new value,
    // updating the parent object/array accordingly.
    public void ReplaceWith<T>(T value);

    // Asynchronously parses a stream as UTF-8 encoded data
    // representing a single JSON value into a JsonNode.
    public static Task<JsonNode?> ParseAsync(
        Stream utf8Json,
        JsonNodeOptions? nodeOptions = null,
        JsonDocumentOptions documentOptions = default,
        CancellationToken cancellationToken = default);
}

public partial class JsonArray
{
    // Returns an IEnumerable<T> view of the current array.
    public IEnumerable<T> GetValues<T>();
}
```

Non-public members

You can opt non-public members into the serialization contract for a given type using JsonIncludeAttribute and JsonConstructorAttribute attribute annotations.

C#

```
public static void NonPublicMembers()
{
    string json = JsonSerializer.Serialize(new MyPoco(42));
    Console.WriteLine(json);
    // {"X":42}

    JsonSerializer.Deserialize<MyPoco>(json);
}
```

```
public class MyPoco
{
    [JsonConstructor]
    internal MyPoco(int x) => X = x;

    [JsonInclude]
    internal int X { get; }
}
```

For more information, see Use immutable types and non-public members and accessors.

Streaming deserialization APIs

.NET 8 includes new IAsyncEnumerable<T> streaming deserialization extension methods, for example GetFromJsonAsAsyncEnumerable. Similar methods have existed that return Task<TResult>, for example, HttpClientJsonExtensions.GetFromJsonAsync. The new extension methods invoke streaming APIs and return IAsyncEnumerable<T>.

The following code shows how you might use the new extension methods.

C#

```
public async static void StreamingDeserialization()
{
    const string RequestUri = "https://api.contoso.com/books";
    using var client = new HttpClient();
    IAsyncEnumerable<Book?> books =
client.GetFromJsonAsAsyncEnumerable<Book>(RequestUri);

    await foreach (Book? book in books)
    {
        Console.WriteLine($"Read book '{book?.title}'");
    }
}

public record Book(int id, string title, string author, int publishedYear);
```

WithAddedModifier extension method

The new WithAddedModifier(IJsonTypeInfoResolver, Action<JsonTypeInfo>) extension method lets you easily introduce modifications to the serialization contracts of arbitrary `IJsonTypeInfoResolver` instances.

C#

```
var options = new JsonSerializerOptions
{
    TypeInfoResolver = MyContext.Default
        .WithAddedModifier(static typeInfo =>
        {
            foreach (JsonPropertyInfo prop in typeInfo.Properties)
            {
                prop.Name = prop.Name.ToUpperInvariant();
            }
        })
};
```

New JsonContent.Create overloads

You can now create JsonContent instances using trim-safe or source-generated contracts. The new methods are:

- JsonContent.Create(Object, JsonTypeInfo, MediaTypeHeaderValue)
- JsonContent.Create<T>(T, JsonTypeInfo<T>, MediaTypeHeaderValue)

C#

```
var book = new Book(id: 42, "Title", "Author", publishedYear: 2023);
HttpContent content = JsonContent.Create(book, MyContext.Default.Book);

public record Book(int id, string title, string author, int publishedYear);

[JsonSerializable(typeof(Book))]
public partial class MyContext : JsonSerializerContext
{
}
```

Freeze a JsonSerializerOptions instance

The following new methods let you control when a JsonSerializerOptions instance is frozen:

- JsonSerializerOptions.MakeReadOnly()

  This overload is designed to be trim-safe and will therefore throw an exception in cases where the options instance hasn't been configured with a resolver.

- JsonSerializerOptions.MakeReadOnly(Boolean)

  If you pass `true` to this overload, it populates the options instance with the default reflection resolver if one is missing. This method is marked

`RequiresUnreferenceCode`/`RequiresDynamicCode` and is therefore unsuitable for Native AOT applications.

The new IsReadOnly property lets you check if the options instance is frozen.

Time abstraction

The new TimeProvider class and ITimer interface add *time abstraction* functionality, which allows you to mock time in test scenarios. In addition, you can use the time abstraction to mock Task operations that rely on time progression using Task.Delay and Task.WaitAsync. The time abstraction supports the following essential time operations:

- Retrieve local and UTC time
- Obtain a timestamp for measuring performance
- Create a timer

The following code snippet shows some usage examples.

C#

```
// Get system time.
DateTimeOffset utcNow = TimeProvider.System.GetUtcNow();
DateTimeOffset localNow = TimeProvider.System.GetLocalNow();

TimerCallback callback = s => ((State)s!).Signal();

// Create a timer using the time provider.
ITimer timer = _timeProvider.CreateTimer(
    callback, null, TimeSpan.Zero, Timeout.InfiniteTimeSpan);

// Measure a period using the system time provider.
long providerTimestamp1 = TimeProvider.System.GetTimestamp();
long providerTimestamp2 = TimeProvider.System.GetTimestamp();

TimeSpan period = _timeProvider.GetElapsedTime(providerTimestamp1,
providerTimestamp2);
```

C#

```
// Create a time provider that works with a
// time zone that's different than the local time zone.
private class ZonedTimeProvider(TimeZoneInfo zoneInfo) : TimeProvider()
{
    private readonly TimeZoneInfo _zoneInfo = zoneInfo ??
TimeZoneInfo.Local;

    public override TimeZoneInfo LocalTimeZone => _zoneInfo;
```

```
    public static TimeProvider FromLocalTimeZone(TimeZoneInfo zoneInfo) =>
        new ZonedTimeProvider(zoneInfo);
}
```

UTF8 improvements

If you want to enable writing out a string-like representation of your type to a destination span, implement the new IUtf8SpanFormattable interface on your type. This new interface is closely related to ISpanFormattable, but targets UTF8 and `Span<byte>` instead of UTF16 and `Span<char>`.

IUtf8SpanFormattable has been implemented on all of the primitive types (plus others), with the exact same shared logic whether targeting `string`, `Span<char>`, or `Span<byte>`. It has full support for all formats (including the new "B" binary specifier) and all cultures. This means you can now format directly to UTF8 from `Byte`, `Complex`, `Char`, `DateOnly`, `DateTime`, `DateTimeOffset`, `Decimal`, `Double`, `Guid`, `Half`, `IPAddress`, `IPNetwork`, `Int16`, `Int32`, `Int64`, `Int128`, `IntPtr`, `NFloat`, `SByte`, `Single`, `Rune`, `TimeOnly`, `TimeSpan`, `UInt16`, `UInt32`, `UInt64`, `UInt128`, `UIntPtr`, and `Version`.

New Utf8.TryWrite methods provide a UTF8-based counterpart to the existing MemoryExtensions.TryWrite methods, which are UTF16-based. You can use interpolated string syntax to format a complex expression directly into a span of UTF8 bytes, for example:

C#

```
static bool FormatHexVersion(
    short major,
    short minor,
    short build,
    short revision,
    Span<byte> utf8Bytes,
    out int bytesWritten) =>
    Utf8.TryWrite(
        utf8Bytes,
        CultureInfo.InvariantCulture,
        $"{major:X4}.{minor:X4}.{build:X4}.{revision:X4}",
        out bytesWritten);
```

The implementation recognizes IUtf8SpanFormattable on the format values and uses their implementations to write their UTF8 representations directly to the destination span.

The implementation also utilizes the new Encoding.TryGetBytes(ReadOnlySpan<Char>, Span<Byte>, Int32) method, which together with its Encoding.TryGetChars(ReadOnlySpan<Byte>, Span<Char>, Int32) counterpart, supports encoding and decoding into a destination span. If the span isn't long enough to hold the resulting state, the methods return `false` rather than throwing an exception.

Methods for working with randomness

The System.Random and System.Security.Cryptography.RandomNumberGenerator types introduce two new methods for working with randomness.

GetItems<T>()

The new System.Random.GetItems and System.Security.Cryptography.RandomNumberGenerator.GetItems methods let you randomly choose a specified number of items from an input set. The following example shows how to use `System.Random.GetItems<T>()` (on the instance provided by the Random.Shared property) to randomly insert 31 items into an array. This example could be used in a game of "Simon" where players must remember a sequence of colored buttons.

C#

```
private static ReadOnlySpan<Button> s_allButtons = new[]
{
    Button.Red,
    Button.Green,
    Button.Blue,
    Button.Yellow,
};

// ...

Button[] thisRound = Random.Shared.GetItems(s_allButtons, 31);
// Rest of game goes here ...
```

Shuffle<T>()

The new Random.Shuffle and RandomNumberGenerator.Shuffle<T>(Span<T>) methods let you randomize the order of a span. These methods are useful for reducing training bias in machine learning (so the first thing isn't always training, and the last thing always test).

C#

```
YourType[] trainingData = LoadTrainingData();
Random.Shared.Shuffle(trainingData);

IDataView sourceData = mlContext.Data.LoadFromEnumerable(trainingData);

DataOperationsCatalog.TrainTestData split =
mlContext.Data.TrainTestSplit(sourceData);
model = chain.Fit(split.TrainSet);

IDataView predictions = model.Transform(split.TestSet);
// ...
```

Performance-focused types

.NET 8 introduces several new types aimed at improving app performance.

- The new System.Collections.Frozen namespace includes the collection types FrozenDictionary<TKey,TValue> and FrozenSet<T>. These types don't allow any changes to keys and values once a collection created. That requirement allows faster read operations (for example, `TryGetValue()`). These types are particularly useful for collections that are populated on first use and then persisted for the duration of a long-lived service, for example:

  C#

  ```
  private static readonly FrozenDictionary<string, bool>
  s_configurationData =
      LoadConfigurationData().ToFrozenDictionary(optimizeForReads: true);

  // ...
  if (s_configurationData.TryGetValue(key, out bool setting) && setting)
  {
      Process();
  }
  ```

- Methods like MemoryExtensions.IndexOfAny look for the first occurrence of *any value in the passed collection*. The new System.Buffers.SearchValues<T> type is designed to be passed to such methods. Correspondingly, .NET 8 adds new overloads of methods like MemoryExtensions.IndexOfAny that accept an instance of the new type. When you create an instance of SearchValues<T>, all the data that's necessary to optimize subsequent searches is derived *at that time*, meaning the work is done up front.

- The new System.Text.CompositeFormat type is useful for optimizing format strings that aren't known at compile time (for example, if the format string is loaded from a resource file). A little extra time is spent up front to do work such as parsing the string, but it saves the work from being done on each use.

  C#

  ```
  private static readonly CompositeFormat s_rangeMessage =
      CompositeFormat.Parse(LoadRangeMessageResource());

  // ...
  static string GetMessage(int min, int max) =>
      string.Format(CultureInfo.InvariantCulture, s_rangeMessage, min,
  max);
  ```

- New System.IO.Hashing.XxHash3 and System.IO.Hashing.XxHash128 types provide implementations of the fast XXH3 and XXH128 hash algorithms.

System.Numerics and System.Runtime.Intrinsics

This section covers improvements to the System.Numerics and System.Runtime.Intrinsics namespaces.

- Vector256<T>, Matrix3x2, and Matrix4x4 have improved hardware acceleration on .NET 8. For example, Vector256<T> was reimplemented to internally be `2x Vector128<T>` operations, where possible. This allows partial acceleration of some functions when `Vector128.IsHardwareAccelerated == true` but `Vector256.IsHardwareAccelerated == false`, such as on Arm64.
- Hardware intrinsics are now annotated with the `ConstExpected` attribute. This ensures that users are aware when the underlying hardware expects a constant and therefore when a non-constant value may unexpectedly hurt performance.
- The Lerp(TSelf, TSelf, TSelf) `Lerp` API has been added to IFloatingPointIeee754<TSelf> and therefore to `float` (Single), `double` (Double), and Half. This API allows a linear interpolation between two values to be performed efficiently and correctly.

Vector512 and AVX-512

.NET Core 3.0 expanded SIMD support to include the platform-specific hardware intrinsics APIs for x86/x64. .NET 5 added support for Arm64 and .NET 7 added the cross-platform hardware intrinsics. .NET 8 furthers SIMD support by introducing Vector512<T> and support for Intel Advanced Vector Extensions 512 (AVX-512) instructions.

Specifically, .NET 8 includes support for the following key features of AVX-512:

- 512-bit vector operations
- Additional 16 SIMD registers
- Additional instructions available for 128-bit, 256-bit, and 512-bit vectors

If you have hardware that supports the functionality, then Vector512.IsHardwareAccelerated now reports `true`.

.NET 8 also adds several platform-specific classes under the System.Runtime.Intrinsics.X86 namespace:

- Avx512F (foundational)
- Avx512BW (byte and word)
- Avx512CD (conflict detection)
- Avx512DQ (doubleword and quadword)
- Avx512Vbmi (vector byte manipulation instructions)

These classes follow the same general shape as other instruction set architectures (ISAs) in that they expose an IsSupported property and a nested Avx512F.X64 class for instructions available only to 64-bit processes. Additionally, each class has a nested Avx512F.VL class that exposes the `Avx512VL` (vector length) extensions for the corresponding instruction set.

Even if you don't explicitly use `Vector512`-specific or `Avx512F`-specific instructions in your code, you'll likely still benefit from the new AVX-512 support. The JIT can take advantage of the additional registers and instructions implicitly when using Vector128<T> or Vector256<T>. The base class library uses these hardware intrinsics internally in most operations exposed by Span<T> and ReadOnlySpan<T> and in many of the math APIs exposed for the primitive types.

Data validation

The System.ComponentModel.DataAnnotations namespace includes new data validation attributes intended for validation scenarios in cloud-native services. While the pre-existing `DataAnnotations` validators are geared towards typical UI data-entry validation, such as fields on a form, the new attributes are designed to validate non-user-entry data, such as configuration options. In addition to the new attributes, new properties were added to the RangeAttribute and RequiredAttribute types.

New API	Description
RangeAttribute.MinimumIsExclusive RangeAttribute.MaximumIsExclusive	Specifies whether bounds are included in the allowable range.
System.ComponentModel.DataAnnotations.LengthAttribute	Specifies both lower and upper bounds for strings or collections. For example, `[Length(10, 20)]` requires at least 10 elements and at most 20 elements in a collection.
System.ComponentModel.DataAnnotations.Base64StringAttribute	Validates that a string is a valid Base64 representation.
System.ComponentModel.DataAnnotations.AllowedValuesAttribute System.ComponentModel.DataAnnotations.DeniedValuesAttribute	Specify allow lists and deny lists, respectively. For example, `[AllowedValues("apple", "banana", "mango")]`.

Metrics

New APIs let you attach key-value pair tags to Meter and Instrument objects when you create them. Aggregators of published metric measurements can use the tags to differentiate the aggregated values.

C#

```
var options = new MeterOptions("name")
{
    Version = "version",
    // Attach these tags to the created meter.
    Tags = new TagList()
    {
        { "MeterKey1", "MeterValue1" },
        { "MeterKey2", "MeterValue2" }
    }
};

Meter meter = meterFactory!.Create(options);

Counter<int> counterInstrument = meter.CreateCounter<int>(
    "counter", null, null, new TagList() { { "counterKey1", "counterValue1"
} }
);
counterInstrument.Add(1);
```

The new APIs include:

- MeterOptions
- Meter(MeterOptions)
- CreateCounter<T>(String, String, String, IEnumerable<KeyValuePair<String,Object>>)

Cryptography

.NET 8 adds support for the SHA-3 hashing primitives. (SHA-3 is currently supported by Linux with OpenSSL 1.1.1 or later and Windows 11 Build 25324 or later.) APIs where SHA-2 is available now offer a SHA-3 compliment. This includes `SHA3_256`, `SHA3_384`, and `SHA3_512` for hashing; `HMACSHA3_256`, `HMACSHA3_384`, and `HMACSHA3_512` for HMAC; `HashAlgorithmName.SHA3_256`, `HashAlgorithmName.SHA3_384`, and `HashAlgorithmName.SHA3_512` for hashing where the algorithm is configurable; and `RSAEncryptionPadding.OaepSHA3_256`, `RSAEncryptionPadding.OaepSHA3_384`, and `RSAEncryptionPadding.OaepSHA3_512` for RSA OAEP encryption.

The following example shows how to use the APIs, including the `SHA3_256.IsSupported` property to determine if the platform supports SHA-3.

C#

```
// Hashing example
if (SHA3_256.IsSupported)
{
    byte[] hash = SHA3_256.HashData(dataToHash);
}
else
{
    // ...
}

// Signing example
if (SHA3_256.IsSupported)
{
     using ECDsa ec = ECDsa.Create(ECCurve.NamedCurves.nistP256);
     byte[] signature = ec.SignData(dataToBeSigned,
HashAlgorithmName.SHA3_256);
}
else
{
    // ...
}
```

SHA-3 support is currently aimed at supporting cryptographic primitives. Higher-level constructions and protocols aren't expected to fully support SHA-3 initially. These protocols include X.509 certificates, SignedXml, and COSE.

Networking

Support for HTTPS proxy

Until now, the proxy types that HttpClient supported all allowed a "man-in-the-middle" to see which site the client is connecting to, even for HTTPS URIs. HttpClient now supports *HTTPS proxy*, which creates an encrypted channel between the client and the proxy so all requests can be handled with full privacy.

To enable HTTPS proxy, set the `all_proxy` environment variable, or use the WebProxy class to control the proxy programmatically.

Unix: `export all_proxy=https://x.x.x.x:3218` Windows: `set all_proxy=https://x.x.x.x:3218`

You can also use the WebProxy class to control the proxy programmatically.

Stream-based ZipFile methods

.NET 8 includes new overloads of ZipFile.CreateFromDirectory that allow you to collect all the files included in a directory and zip them, then store the resulting zip file into the provided stream. Similarly, new ZipFile.ExtractToDirectory overloads let you provide a stream containing a zipped file and extract its contents into the filesystem. These are the new overloads:

C#

```
namespace System.IO.Compression;

public static partial class ZipFile
{
    public static void CreateFromDirectory(
        string sourceDirectoryName, Stream destination);

    public static void CreateFromDirectory(
        string sourceDirectoryName,
        Stream destination,
        CompressionLevel compressionLevel,
        bool includeBaseDirectory);

    public static void CreateFromDirectory(
```

```
        string sourceDirectoryName,
        Stream destination,
        CompressionLevel compressionLevel,
        bool includeBaseDirectory,
    Encoding? entryNameEncoding);

    public static void ExtractToDirectory(
        Stream source, string destinationDirectoryName) { }

    public static void ExtractToDirectory(
        Stream source, string destinationDirectoryName, bool overwriteFiles)
{ }

    public static void ExtractToDirectory(
        Stream source, string destinationDirectoryName, Encoding?
entryNameEncoding) { }

    public static void ExtractToDirectory(
        Stream source, string destinationDirectoryName, Encoding?
entryNameEncoding, bool overwriteFiles) { }
}
```

These new APIs can be useful when disk space is constrained, because they avoid having to use the disk as an intermediate step.

Extension libraries

This section contains the following subtopics:

- Options validation
- LoggerMessageAttribute constructors
- Extensions metrics
- Hosted lifecycle services
- Keyed DI services
- System.Numerics.Tensors.TensorPrimitives

Keyed DI services

Keyed dependency injection (DI) services provide a means for registering and retrieving DI services using keys. By using keys, you can scope how you register and consume services. These are some of the new APIs:

- The IKeyedServiceProvider interface.
- The ServiceKeyAttribute attribute, which can be used to inject the key that was used for registration/resolution in the constructor.

- The FromKeyedServicesAttribute attribute, which can be used on service constructor parameters to specify which keyed service to use.
- Various new extension methods for IServiceCollection to support keyed services, for example, ServiceCollectionServiceExtensions.AddKeyedScoped.
- The ServiceProvider implementation of IKeyedServiceProvider.

The following example shows you how to use keyed DI services.

C#

```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
builder.Services.AddSingleton<BigCacheConsumer>();
builder.Services.AddSingleton<SmallCacheConsumer>();
builder.Services.AddKeyedSingleton<ICache, BigCache>("big");
builder.Services.AddKeyedSingleton<ICache, SmallCache>("small");
WebApplication app = builder.Build();
app.MapGet("/big", (BigCacheConsumer data) => data.GetData());
app.MapGet("/small", (SmallCacheConsumer data) => data.GetData());
app.MapGet("/big-cache", ([FromKeyedServices("big")] ICache cache) =>
cache.Get("data"));
app.MapGet("/small-cache", (HttpContext httpContext) =>
httpContext.RequestServices.GetRequiredKeyedService<ICache>
("small").Get("data"));
app.Run();

class BigCacheConsumer([FromKeyedServices("big")] ICache cache)
{
    public object? GetData() => cache.Get("data");
}

class SmallCacheConsumer(IServiceProvider serviceProvider)
{
    public object? GetData() =>
serviceProvider.GetRequiredKeyedService<ICache>("small").Get("data");
}

public interface ICache
{
    object Get(string key);
}

public class BigCache : ICache
{
    public object Get(string key) => $"Resolving {key} from big cache.";
}

public class SmallCache : ICache
{
    public object Get(string key) => $"Resolving {key} from small cache.";
}
```

For more information, see dotnet/runtime#64427.

Hosted lifecycle services

Hosted services now have more options for execution during the application lifecycle. IHostedService provided `StartAsync` and `StopAsync`, and now IHostedLifecycleService provides these additional methods:

- StartingAsync(CancellationToken)
- StartedAsync(CancellationToken)
- StoppingAsync(CancellationToken)
- StoppedAsync(CancellationToken)

These methods run before and after the existing points respectively.

The following example shows how to use the new APIs.

C#

```
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

internal class HostedLifecycleServices
{
    public async static void RunIt()
    {
        IHostBuilder hostBuilder = new HostBuilder();
        hostBuilder.ConfigureServices(services =>
        {
            services.AddHostedService<MyService>();
        });

        using (IHost host = hostBuilder.Build())
        {
            await host.StartAsync();
        }
    }

    public class MyService : IHostedLifecycleService
    {
        public Task StartingAsync(CancellationToken cancellationToken) => /*
add logic here */ Task.CompletedTask;
        public Task StartAsync(CancellationToken cancellationToken) => /*
add logic here */ Task.CompletedTask;
        public Task StartedAsync(CancellationToken cancellationToken) => /*
add logic here */ Task.CompletedTask;
        public Task StopAsync(CancellationToken cancellationToken) => /* add
logic here */ Task.CompletedTask;
```

```
        public Task StoppedAsync(CancellationToken cancellationToken) => /*
add logic here */ Task.CompletedTask;
        public Task StoppingAsync(CancellationToken cancellationToken) => /*
add logic here */ Task.CompletedTask;
    }
}
```

For more information, see dotnet/runtime#86511.

Options validation

Source generator

To reduce startup overhead and improve the validation feature set, we've introduced a source-code generator that implements the validation logic. The following code shows example models and validator classes.

C#

```
public class FirstModelNoNamespace
{
    [Required]
    [MinLength(5)]
    public string P1 { get; set; } = string.Empty;

    [Microsoft.Extensions.Options.ValidateObjectMembers(
        typeof(SecondValidatorNoNamespace))]
    public SecondModelNoNamespace? P2 { get; set; }
}

public class SecondModelNoNamespace
{
    [Required]
    [MinLength(5)]
    public string P4 { get; set; } = string.Empty;
}

[OptionsValidator]
public partial class FirstValidatorNoNamespace
    : IValidateOptions<FirstModelNoNamespace>
{
}

[OptionsValidator]
public partial class SecondValidatorNoNamespace
    : IValidateOptions<SecondModelNoNamespace>
{
}
```

If your app uses dependency injection, you can inject the validation as shown in the following example code.

```C#
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.Configure<FirstModelNoNamespace>(
    builder.Configuration.GetSection("some string"));

builder.Services.AddSingleton<
    IValidateOptions<FirstModelNoNamespace>, FirstValidatorNoNamespace>();
builder.Services.AddSingleton<
    IValidateOptions<SecondModelNoNamespace>, SecondValidatorNoNamespace>();
```

ValidateOptionsResultBuilder type

.NET 8 introduces the ValidateOptionsResultBuilder type to facilitate the creation of a ValidateOptionsResult object. Importantly, this builder allows for the accumulation of multiple errors. Previously, creating the ValidateOptionsResult object that's required to implement IValidateOptions<TOptions>.Validate(String, TOptions) was difficult and sometimes resulted in layered validation errors. If there were multiple errors, the validation process often stopped at the first error.

The following code snippet shows an example usage of ValidateOptionsResultBuilder.

```C#
ValidateOptionsResultBuilder builder = new();
builder.AddError("Error: invalid operation code");
builder.AddResult(ValidateOptionsResult.Fail("Invalid request parameters"));
builder.AddError("Malformed link", "Url");

// Build ValidateOptionsResult object has accumulating multiple errors.
ValidateOptionsResult result = builder.Build();

// Reset the builder to allow using it in new validation operation.
builder.Clear();
```

LoggerMessageAttribute constructors

LoggerMessageAttribute now offers additional constructor overloads. Previously, you had to choose either the parameterless constructor or the constructor that required all of the parameters (event ID, log level, and message). The new overloads offer greater

flexibility in specifying the required parameters with reduced code. If you don't supply an event ID, the system generates one automatically.

```
C#
```

```
public LoggerMessageAttribute(LogLevel level, string message);
public LoggerMessageAttribute(LogLevel level);
public LoggerMessageAttribute(string message);
```

Extensions metrics

IMeterFactory interface

You can register the new IMeterFactory interface in dependency injection (DI) containers and use it to create Meter objects in an isolated manner.

Register the IMeterFactory to the DI container using the default meter factory implementation:

```
C#
```

```
// 'services' is the DI IServiceCollection.
services.AddMetrics();
```

Consumers can then obtain the meter factory and use it to create a new Meter object.

```
C#
```

```
IMeterFactory meterFactory =
serviceProvider.GetRequiredService<IMeterFactory>();

MeterOptions options = new MeterOptions("MeterName")
{
    Version = "version",
};

Meter meter = meterFactory.Create(options);
```

MetricCollector<T> class

The new MetricCollector<T> class lets you record metric measurements along with timestamps. Additionally, the class offers the flexibility to use a time provider of your choice for accurate timestamp generation.

C#

```
const string CounterName = "MyCounter";
DateTimeOffset now = DateTimeOffset.Now;

var timeProvider = new FakeTimeProvider(now);
using var meter = new Meter(Guid.NewGuid().ToString());
Counter<long> counter = meter.CreateCounter<long>(CounterName);
using var collector = new MetricCollector<long>(counter, timeProvider);

Assert.IsNull(collector.LastMeasurement);

counter.Add(3);

// Verify the update was recorded.
Assert.AreEqual(counter, collector.Instrument);
Assert.IsNotNull(collector.LastMeasurement);

Assert.AreSame(collector.GetMeasurementSnapshot().Last(),
collector.LastMeasurement);
Assert.AreEqual(3, collector.LastMeasurement.Value);
Assert.AreEqual(now, collector.LastMeasurement.Timestamp);
```

System.Numerics.Tensors.TensorPrimitives

The updated System.Numerics.Tensors NuGet package includes APIs in the new TensorPrimitives namespace that add support for tensor operations. The tensor primitives optimize data-intensive workloads like those of AI and machine learning.

AI workloads like semantic search and retrieval-augmented generation (RAG) extend the natural language capabilities of large language models such as ChatGPT by augmenting prompts with relevant data. For these workloads, operations on vectors—like *cosine similarity* to find the most relevant data to answer a question—are crucial. The System.Numerics.Tensors.TensorPrimitives package provides APIs for vector operations, meaning you don't need to take an external dependency or write your own implementation.

This package replaces the System.Numerics.Tensors package.

For more information, see the Announcing .NET 8 RC 2 blog post.

See also

- What's new in .NET 8

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

What's new in the SDK and tooling for .NET 8

Article • 02/08/2024

This article describes new features in the .NET SDK and tooling for .NET 8.

SDK

This section contains the following subtopics:

- CLI-based project evaluation
- Terminal build output
- Simplified output paths
- 'dotnet workload clean' command
- 'dotnet publish' and 'dotnet pack' assets
- Template engine
- Source Link
- Source-build SDK

CLI-based project evaluation

MSBuild includes a new feature that makes it easier to incorporate data from MSBuild into your scripts or tools. The following new flags are available for CLI commands such as dotnet publish to obtain data for use in CI pipelines and elsewhere.

Expand table

Flag	Description
`--getProperty:<PROPERTYNAME>`	Retrieves the MSBuild property with the specified name.
`--getItem:<ITEMTYPE>`	Retrieves MSBuild items of the specified type.
`--getTargetResults:<TARGETNAME>`	Retrieves the outputs from running the specified target.

Values are written to the standard output. Multiple or complex values are output as JSON, as shown in the following examples.

.NET CLI

```
>dotnet publish --getProperty:OutputPath
```

```
bin\Release\net8.0\
```

.NET CLI

```
>dotnet publish -p PublishProfile=DefaultContainer --
getProperty:GeneratedContainerDigest --
getProperty:GeneratedContainerConfiguration
{
  "Properties": {
    "GeneratedContainerDigest":
"sha256:ef880a503bbabcb84bbb6a1aa9b41b36dc1ba08352e7cd91c0993646675174c4",
    "GeneratedContainerConfiguration": "{\u0022config\u0022:
{\u0022ExposedPorts\u0022:{\u00228080/tcp\u0022:{}},\u0022Labels\u0022...}}"
  }
}
```

.NET CLI

```
>dotnet publish -p PublishProfile=DefaultContainer --
getItem:ContainerImageTags
{
  "Items": {
    "ContainerImageTags": [
      {
        "Identity": "latest",
        ...
    ]
  }
}
```

Terminal build output

`dotnet build` has a new option to produce more modernized build output. This *terminal logger* output groups errors with the project they came from, better differentiates the different target frameworks for multi-targeted projects, and provides real-time information about what the build is doing. To opt into the new output, use the `--tl` option. For more information about this option, see dotnet build options.

Simplified output paths

.NET 8 introduces an option to simplify the output path and folder structure for build outputs. Previously, .NET apps produced a deep and complex set of output paths for different build artifacts. The new, simplified output path structure gathers all build outputs into a common location, which makes it easier for tooling to anticipate.

For more information, see Artifacts output layout.

`dotnet workload clean` command

.NET 8 introduces a new command to clean up workload packs that might be left over through several .NET SDK or Visual Studio updates. If you encounter issues when managing workloads, consider using `workload clean` to safely restore to a known state before trying again. The command has two modes:

- `dotnet workload clean`

  Runs workload garbage collection for file-based or MSI-based workloads, which cleans up orphaned packs. Orphaned packs are from uninstalled versions of the .NET SDK or packs where installation records for the pack no longer exist.

  If Visual Studio is installed, the command also lists any workloads that you should clean up manually using Visual Studio.

- `dotnet workload clean --all`

  This mode is more aggressive and cleans every pack on the machine that's of the current SDK workload installation type (and that's not from Visual Studio). It also removes all workload installation records for the running .NET SDK feature band and below.

`dotnet publish` and `dotnet pack` assets

Since the dotnet publish and dotnet pack commands are intended to produce production assets, they now produce `Release` assets by default.

The following output shows the different behavior between `dotnet build` and `dotnet publish`, and how you can revert to publishing `Debug` assets by setting the `PublishRelease` property to `false`.

Console

```
/app# dotnet new console
/app# dotnet build
  app -> /app/bin/Debug/net8.0/app.dll
/app# dotnet publish
  app -> /app/bin/Release/net8.0/app.dll
  app -> /app/bin/Release/net8.0/publish/
/app# dotnet publish -p:PublishRelease=false
```

```
  app -> /app/bin/Debug/net8.0/app.dll
  app -> /app/bin/Debug/net8.0/publish/
```

For more information, see 'dotnet pack' uses Release config and 'dotnet publish' uses Release config.

`dotnet restore` security auditing

Starting in .NET 8, you can opt into security checks for known vulnerabilities when dependency packages are restored. This auditing produces a report of security vulnerabilities with the affected package name, the severity of the vulnerability, and a link to the advisory for more details. When you run `dotnet add` or `dotnet restore`, warnings NU1901-NU1904 will appear for any vulnerabilities that are found. For more information, see Audit for security vulnerabilities.

Template engine

The template engine provides a more secure experience in .NET 8 by integrating some of NuGet's security-related features. The improvements include:

- Prevent downloading packages from `http://` feeds by default. For example, the following command will fail to install the template package because the source URL doesn't use HTTPS.

  `dotnet new install console --add-source "http://pkgs.dev.azure.com/dnceng/public/_packaging/dotnet-public/nuget/v3/index.json"`

  You can override this limitation by using the `--force` flag.

- For `dotnet new`, `dotnet new install`, and `dotnet new update`, check for known vulnerabilities in the template package. If vulnerabilities are found and you wish to proceed, you must use the `--force` flag.

- For `dotnet new`, provide information about the template package owner. Ownership is verified by the NuGet portal and can be considered a trustworthy characteristic.

- For `dotnet search` and `dotnet uninstall`, indicate whether a template is installed from a package that's "trusted"—that is, it uses a reserved prefix.

Source Link

Source Link is now included in the .NET SDK. The goal is that by bundling Source Link into the SDK, instead of requiring a separate `<PackageReference>` for the package, more packages will include this information by default. That information will improve the IDE experience for developers.

> **Note**
>
> As a side effect of this change, commit information is included in the `InformationalVersion` value of built libraries and applications, even those that target .NET 7 or an earlier version. For more information, see **Source Link included in the .NET SDK**.

Source-build SDK

The Linux distribution-built (source-build) SDK now has the capability to build self-contained applications using the source-build runtime packages. The distribution-specific runtime package is bundled with the source-build SDK. During self-contained deployment, this bundled runtime package will be referenced, thereby enabling the feature for users.

Native AOT support

The option to publish as Native AOT was first introduced in .NET 7. Publishing an app with Native AOT creates a fully self-contained version of your app that doesn't need a runtime—everything is included in a single file. .NET 8 brings the following improvements to Native AOT publishing:

- Adds support for the x64 and Arm64 architectures on *macOS*.
- Reduces the sizes of Native AOT apps on Linux by up to 50%. The following table shows the size of a "Hello World" app published with Native AOT that includes the entire .NET runtime on .NET 7 vs. .NET 8:

Expand table

Operating system	.NET 7	.NET 8
Linux x64 (with `-p:StripSymbols=true`)	3.76 MB	1.84 MB
Windows x64	2.85 MB	1.77 MB

- Lets you specify an optimization preference: size or speed. By default, the compiler chooses to generate fast code while being mindful of the size of the application. However, you can use the `<OptimizationPreference>` MSBuild property to optimize specifically for one or the other. For more information, see Optimize AOT deployments.

Console app template

The default console app template now includes support for AOT out-of-the-box. To create a project that's configured for AOT compilation, just run `dotnet new console --aot`. The project configuration added by `--aot` has three effects:

- Generates a native self-contained executable with Native AOT when you publish the project, for example, with `dotnet publish` or Visual Studio.
- Enables compatibility analyzers for trimming, AOT, and single file. These analyzers alert you to potentially problematic parts of your project (if there are any).
- Enables debug-time emulation of AOT so that when you debug your project without AOT compilation, you get a similar experience to AOT. For example, if you use System.Reflection.Emit in a NuGet package that wasn't annotated for AOT (and therefore was missed by the compatibility analyzer), the emulation means you won't have any surprises when you try to publish the project with AOT.

Target iOS-like platforms with Native AOT

.NET 8 starts the work to enable Native AOT support for iOS-like platforms. You can now build and run .NET iOS and .NET MAUI applications with Native AOT on the following platforms:

- `ios`
- `iossimulator`
- `maccatalyst`
- `tvos`
- `tvossimulator`

Preliminary testing shows that app size on disk decreases by about 35% for .NET iOS apps that use Native AOT instead of Mono. App size on disk for .NET MAUI iOS apps decreases up to 50%. Additionally, the startup time is also faster. .NET iOS apps have about 28% faster startup time, while .NET MAUI iOS apps have about 50% better startup performance compared to Mono. The .NET 8 support is experimental and only the first step for the feature as a whole. For more information, see the .NET 8 Performance Improvements in .NET MAUI blog post.

Native AOT support is available as an opt-in feature intended for app deployment; Mono is still the default runtime for app development and deployment. To build and run a .NET MAUI application with Native AOT on an iOS device, use `dotnet workload install maui` to install the .NET MAUI workload and `dotnet new maui -n HelloMaui` to create the app. Then, set the MSBuild property `PublishAot` to `true` in the project file.

XML

```xml
<PropertyGroup>
  <PublishAot>true</PublishAot>
</PropertyGroup>
```

When you set the required property and run `dotnet publish` as shown in the following example, the app will be deployed by using Native AOT.

.NET CLI

```dotnetcli
dotnet publish -f net8.0-ios -c Release -r ios-arm64  /t:Run
```

Limitations

Not all iOS features are compatible with Native AOT. Similarly, not all libraries commonly used in iOS are compatible with NativeAOT. And in addition to the existing limitations of Native AOT deployment, the following list shows some of the other limitations when targeting iOS-like platforms:

- Using Native AOT is only enabled during app deployment (`dotnet publish`).
- Managed code debugging is only supported with Mono.
- Compatibility with the .NET MAUI framework is limited.

AOT compilation for Android apps

To decrease app size, .NET and .NET MAUI apps that target Android use *profiled* ahead-of-time (AOT) compilation mode when they're built in Release mode. Profiled AOT compilation affects fewer methods than regular AOT compilation. .NET 8 introduces the `<AndroidStripILAfterAOT>` property that lets you opt-in to further AOT compilation for Android apps to decrease app size even more.

XML

```xml
<PropertyGroup>
  <AndroidStripILAfterAOT>true</AndroidStripILAfterAOT>
```

```
</PropertyGroup>
```

By default, setting `AndroidStripILAfterAOT` to `true` overrides the default `AndroidEnableProfiledAot` setting, allowing (nearly) all methods that were AOT-compiled to be trimmed. You can also use profiled AOT and IL stripping together by explicitly setting both properties to `true`:

XML

```
<PropertyGroup>
  <AndroidStripILAfterAOT>true</AndroidStripILAfterAOT>
  <AndroidEnableProfiledAot>true</AndroidEnableProfiledAot>
</PropertyGroup>
```

Cross-built Windows apps

When you build apps that target Windows on non-Windows platforms, the resulting executable is now updated with any specified Win32 resources—for example, application icon, manifest, version information.

Previously, applications had to be built on Windows in order to have such resources. Fixing this gap in cross-building support has been a popular request, as it was a significant pain point affecting both infrastructure complexity and resource usage.

.NET on Linux

Minimum support baselines for Linux

The minimum support baselines for Linux have been updated for .NET 8. .NET is built targeting Ubuntu 16.04, for all architectures. That's primarily important for defining the minimum `glibc` version for .NET 8. .NET 8 will fail to start on distro versions that include an older glibc, such as Ubuntu 14.04 or Red Hat Enterprise Linux 7.

For more information, see Red Hat Enterprise Linux Family support.

Build your own .NET on Linux

In previous .NET versions, you could build .NET from source, but it required you to create a "source tarball" from the dotnet/installer repo commit that corresponded to a release. In .NET 8, that's no longer necessary and you can build .NET on Linux directly

from the dotnet/dotnet repository. That repo uses dotnet/source-build to build .NET runtimes, tools, and SDKs. This is the same build that Red Hat and Canonical use to build .NET.

Building in a container is the easiest approach for most people, since the `dotnet-buildtools/prereqs` container images contain all the required dependencies. For more information, see the build instructions.

NuGet signature verification

Starting in .NET 8, NuGet verifies signed packages on Linux by default. NuGet continues to verify signed packages on Windows as well.

Most users shouldn't notice the verification. However, if you have an existing root certificate bundle located at */etc/pki/ca-trust/extracted/pem/objsign-ca-bundle.pem*, you may see trust failures accompanied by warning NU3042.

You can opt out of verification by setting the environment variable `DOTNET_NUGET_SIGNATURE_VERIFICATION` to `false`.

Code analysis

.NET 8 includes several new code analyzers and fixers to help verify that you're using .NET library APIs correctly and efficiently. The following table summarizes the new analyzers.

Expand table

Rule ID	Category	Description
CA1856	Performance	Fires when the ConstantExpectedAttribute attribute is not applied correctly on a parameter.
CA1857	Performance	Fires when a parameter is annotated with ConstantExpectedAttribute but the provided argument isn't a constant.
CA1858	Performance	To determine whether a string starts with a given prefix, it's better to call String.StartsWith than to call String.IndexOf and then compare the result with zero.
CA1859	Performance	This rule recommends upgrading the type of specific local variables, fields, properties, method parameters, and method return types

Rule ID	Category	Description
		from interface or abstract types to concrete types when possible. Using concrete types leads to higher quality generated code.
CA1860	Performance	To determine whether a collection type has any elements, it's better to use `Length`, `Count`, or `IsEmpty` than to call Enumerable.Any.
CA1861	Performance	Constant arrays passed as arguments aren't reused when called repeatedly, which implies a new array is created each time. To improve performance, consider extracting the array to a static readonly field.
CA1865-CA1867	Performance	The char overload is a better-performing overload for a string with a single char.
CA2021	Reliability	Enumerable.Cast<TResult>(IEnumerable) and Enumerable.OfType<TResult>(IEnumerable) require compatible types to function correctly. Widening and user-defined conversions aren't supported with generic types.
CA1510-CA1513	Maintainability	Throw helpers are simpler and more efficient than an `if` block constructing a new exception instance. These four analyzers were created for the following exceptions: ArgumentNullException, ArgumentException, ArgumentOutOfRangeException and ObjectDisposedException.

Diagnostics

C# Hot Reload supports modifying generics

Starting in .NET 8, C# Hot Reload supports modifying generic types and generic methods. When you debug console, desktop, mobile, or WebAssembly applications with Visual Studio, you can apply changes to generic classes and generic methods in C# code or Razor pages. For more information, see the full list of edits supported by Roslyn.

See also

- What's new in .NET 8

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

What's new in containers for .NET 8

Article • 02/08/2024

This article describes new features in containers for .NET 8.

Container images

The following changes have been made to .NET container images for .NET 8:

- Generated-image defaults
- Debian 12
- Non-root user
- Chiseled Ubuntu images
- Build multi-platform container images
- ASP.NET composite images

Generated-image defaults

The new non-root capability of the Microsoft .NET containers is now the default, which helps your apps stay secure-by-default. Change this default at any time by setting your own `ContainerUser`.

The default container tag is now `latest`. This default is in line with other tooling in the containers space and makes containers easier to use in inner development loops.

Debian 12

The container images now use Debian 12 (Bookworm). Debian is the default Linux distro in the .NET container images.

Non-root user

Images include a `non-root` user. This user makes the images `non-root` capable. To run as `non-root`, add the following line at the end of your Dockerfile (or a similar instruction in your Kubernetes manifests):

Dockerfile

```
USER app
```

.NET 8 adds an environment variable for the UID for the `non-root` user, which is 64198. This environment variable is useful for the Kubernetes `runAsNonRoot` test, which requires that the container user be set via UID and not by name. This dockerfile shows an example usage.

The default port also changed from port `80` to `8080`. To support this change, a new environment variable `ASPNETCORE_HTTP_PORTS` is available to make it easier to change ports. The variable accepts a list of ports, which is simpler than the format required by `ASPNETCORE_URLS`. If you change the port back to port `80` using one of these variables, you can't run as `non-root`.

Chiseled Ubuntu images

Chiseled Ubuntu images are available for .NET 8. Chiseled images have a reduced attacked surface because they're ultra-small, have no package manager or shell, and are `non-root`. This type of image is for developers who want the benefit of appliance-style computing. Chiseled images are published to the .NET nightly artifact registry.

Build multi-platform container images

Docker supports using and building multi-platform images that work across multiple environments. .NET 8 introduces a new pattern that enables you to mix and match architectures with the .NET images you build. As an example, if you're using macOS and want to target an x64 cloud service in Azure, you can build the image by using the `--platform` switch as follows:

```
docker build --pull -t app --platform linux/amd64
```

The .NET SDK now supports `$TARGETARCH` values and the `-a` argument on restore. The following code snippet shows an example:

Dockerfile

```
RUN dotnet restore -a $TARGETARCH

# Copy everything else and build app.
COPY aspnetapp/. .
RUN dotnet publish -a $TARGETARCH --self-contained false --no-restore -o
/app
```

For more information, see the Improving multi-platform container support blog post.

ASP.NET composite images

As part of an effort to improve containerization performance, new ASP.NET Docker images are available that have a composite version of the runtime. This composite is built by compiling multiple MSIL assemblies into a single ready-to-run (R2R) output binary. Because these assemblies are embedded into a single image, jitting takes less time, and the startup performance of apps improves. The other big advantage of the composite over the regular ASP.NET image is that the composite images have a smaller size on disk.

There is a caveat to be aware of. Since composites have multiple assemblies embedded into one, they have tighter version coupling. Apps can't use custom versions of framework or ASP.NET binaries.

Composite images are available for the Alpine Linux, Jammy Chiseled, and Mariner Distroless platforms from the `mcr.microsoft.com/dotnet/nightly/aspnet` repo. The tags are listed with the `-composite` suffix on the ASP.NET Docker page.

Container publishing

- Performance and compatibility
- Authentication
- Publish to tar.gz archive

Performance and compatibility

.NET 8 has improved performance for pushing containers to remote registries, especially Azure registries. Speedup comes from pushing layers in one operation and, for registries that don't support atomic uploads, a more reliable chunking mechanism.

These improvements also mean that more registries are supported: Harbor, Artifactory, Quay.io, and Podman.

Authentication

.NET 8 adds support for OAuth token exchange authentication (Azure Managed Identity) when pushing containers to registries. This support means that you can now push to registries like Azure Container Registry without any authentication errors. The following commands show an example publishing flow:

```
Console
```

```
> az acr login -n <your registry name>
> dotnet publish -r linux-x64 -p PublishProfile=DefaultContainer
```

For more information containerizing .NET apps, see Containerize a .NET app with dotnet publish.

Publish to tar.gz archive

Starting in .NET 8, you can create a container directly as a *tar.gz* archive. This feature is useful if your workflow isn't straightforward and requires that you, for example, run a scanning tool over your images before pushing them. Once the archive is created, you can move it, scan it, or load it into a local Docker toolchain.

To publish to an archive, add the `ContainerArchiveOutputPath` property to your `dotnet publish` command, for example:

.NET CLI

```
dotnet publish \
  -p PublishProfile=DefaultContainer \
  -p ContainerArchiveOutputPath=./images/sdk-container-demo.tar.gz
```

You can specify either a folder name or a path with a specific file name.

See also

- What's new in .NET 8

Breaking changes in .NET 8

Article • 12/05/2023

If you're migrating an app to .NET 8, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or Windows Forms.

This article categorizes each breaking change as *binary incompatible* or *source incompatible*, or as a *behavioral change*:

- **Binary incompatible** - When run against the new runtime or component, existing binaries may encounter a breaking change in behavior, such as failure to load or execute, and if so, require recompilation.
- **Source incompatible** - When recompiled using the new SDK or component or to target the new runtime, existing source code may require source changes to compile successfully.
- **Behavioral change** - Existing code and binaries may behave differently at run time. If the new behavior is undesirable, existing code would need to be updated and recompiled.

ASP.NET Core

Expand table

Title	Type of change
ConcurrencyLimiterMiddleware is obsolete	Source incompatible
Custom converters for serialization removed	Behavioral change
ISystemClock is obsolete	Source incompatible
Minimal APIs: IFormFile parameters require anti-forgery checks	Behavioral change
Rate-limiting middleware requires AddRateLimiter	Behavioral change
Security token events return a JsonWebToken	Behavioral change
TrimMode defaults to full for Web SDK projects	Source incompatible

Containers

Expand table

Title	Type of change
'ca-certificates' and 'krb5-libs' packages removed from Alpine images	Binary incompatible
Debian container images upgraded to Debian 12	Binary incompatible/behavioral change
Default ASP.NET Core port changed to 8080	Behavioral change
'libintl' package removed from Alpine images	Behavioral change
Multi-platform container tags are Linux-only	Behavioral change
New 'app' user in Linux images	Behavioral change

Core .NET libraries

Expand table

Title	Type of change
Activity operation name when null	Behavioral change
AnonymousPipeServerStream.Dispose behavior	Behavioral change
API obsoletions with custom diagnostic IDs	Source incompatible
Backslash mapping in Unix file paths	Behavioral change
Base64.DecodeFromUtf8 methods ignore whitespace	Behavioral change
Boolean-backed enum type support removed	Behavioral change
Drive's current directory path enumeration	Behavioral change
Enumerable.Sum throws new OverflowException for some inputs	Behavioral change
FileStream writes when pipe is closed	Behavioral change
GC.GetGeneration might return Int32.MaxValue	Behavioral change
GetFolderPath behavior on Unix	Behavioral change
GetSystemVersion no longer returns ImageRuntimeVersion	Behavioral change
ITypeDescriptorContext nullable annotations	Source incompatible

Title	Type of change
Legacy Console.ReadKey removed	Behavioral change
Method builders generate parameters with HasDefaultValue set to false	Behavioral change
ProcessStartInfo.WindowStyle honored when UseShellExecute is false	Behavioral change
RuntimeIdentifier returns platform for which runtime was built	Behavioral change

Cryptography

Expand table

Title	Type of change	Introduced
AesGcm authentication tag size on macOS	Behavioral change	Preview 1
RSA.EncryptValue and RSA.DecryptValue obsolete	Source incompatible	Preview 1

Deployment

Expand table

Title	Type of change
Host determines RID-specific assets	Binary incompatible/behavioral change
.NET Monitor only includes distroless images	Behavioral change
StripSymbols defaults to true	Behavioral change

Entity Framework Core

Breaking changes in EF Core 8

Extensions

Expand table

Title	Type of change
ActivatorUtilities.CreateInstance behaves consistently	Behavioral change

Title	Type of change
ActivatorUtilities.CreateInstance requires non-null provider	Behavioral change
ConfigurationBinder throws for mismatched value	Behavioral change
ConfigurationManager package no longer references System.Security.Permissions	Source incompatible
DirectoryServices package no longer references System.Security.Permissions	Source incompatible
Empty keys added to dictionary by configuration binder	Behavioral change
HostApplicationBuilderSettings.Args respected by HostApplicationBuilder ctor	Behavioral change
ManagementDateTimeConverter.ToDateTime returns a local time	Behavioral change
System.Formats.Cbor DateTimeOffset formatting change	Behavioral change

Globalization

Expand table

Title	Type of change
Date and time converters honor culture argument	Behavioral change
TwoDigitYearMax default is 2049	Behavioral change

Interop

Expand table

Title	Type of change
CreateObjectFlags.Unwrap only unwraps on target instance	Behavioral change
Custom marshallers require additional members	Source incompatible
IDispatchImplAttribute API is removed	Binary incompatible
JSFunctionBinding implicit public default constructor removed	Binary incompatible
SafeHandle types must have public constructor	Source incompatible

Networking

Expand table

Title	Type of change
SendFile throws NotSupportedException for connectionless sockets	Behavioral change

Reflection

Expand table

Title	Type of change
IntPtr no longer used for function pointer types	Behavioral change

SDK

Expand table

Title	Type of change
CLI console output uses UTF-8	Behavioral change/Source and binary incompatible
Console encoding not UTF-8 after completion	Behavioral change/Binary incompatible
Containers default to use the 'latest' tag	Behavioral change
'dotnet pack' uses Release configuration	Behavioral change/Source incompatible
'dotnet publish' uses Release configuration	Behavioral change/Source incompatible
Duplicate output for -getItem, -getProperty, and -getTargetResult	Behavioral change
Implicit using for System.Net.Http no longer added	Behavioral change/Source incompatible
MSBuild custom derived build events deprecated	Behavioral change
MSBuild respects DOTNET_CLI_UI_LANGUAGE	Behavioral change
Runtime-specific apps not self-contained	Source/binary incompatible
--arch option doesn't imply self-contained	Behavioral change

Title	Type of change
'dotnet restore' produces security vulnerability warnings	Behavioral change
SDK uses a smaller RID graph	Behavioral change/Source incompatible
Source Link included in the .NET SDK	Source incompatible
Trimming may not be used with .NET Standard or .NET Framework	Behavioral change
Unlisted packages not installed by default for .NET tools	Behavioral change
.user file imported in outer builds	Behavioral change
Version requirements for .NET 8 SDK	Source incompatible

Serialization

Expand table

Title	Type of change
BinaryFormatter disabled for most projects	Behavioral change
PublishedTrimmed projects fail reflection-based serialization	Behavioral change
Reflection-based deserializer resolves metadata eagerly	Behavioral change

Windows Forms

Expand table

Title	Type of change
Anchor layout changes	Behavioral change
Certs checked before loading remote images in PictureBox	Behavioral change
DefaultValueAttribute removed from some properties	Behavioral change
ExceptionCollection ctor throws ArgumentException	Behavioral change
Forms scale according to AutoScaleMode	Behavioral change

Title	Type of change
ImageList.ColorDepth default is Depth32Bit	Behavioral change
System.Windows.Extensions doesn't reference System.Drawing.Common	Source incompatible
TableLayoutStyleCollection throws ArgumentException	Behavioral change
Top-level forms scale minimum and maximum size to DPI	Behavioral change
WFDEV002 obsoletion is now an error	Source incompatible

See also

- What's new in .NET 8

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

What's new in .NET 7

Article • 03/09/2023

.NET 7 is the successor to .NET 6 and focuses on being unified, modern, simple, and *fast*. .NET 7 will be supported for 18 months as a standard-term support (STS) release (previously known as a *current* release).

This article lists the new features of .NET 7 and provides links to more detailed information on each.

To find all the .NET articles that have been updated for .NET 7, see .NET docs: What's new for the .NET 7 release.

Performance

Performance is a key focus of .NET 7, and all of its features are designed with performance in mind. In addition, .NET 7 includes the following enhancements aimed purely at performance:

- On-stack replacement (OSR) is a complement to tiered compilation. It allows the runtime to change the code executed by a currently running method in the middle of its execution (that is, while it's "on stack"). Long-running methods can switch to more optimized versions mid-execution.
- Profile-guided optimization (PGO) now works with OSR and is easier to enable (by adding `<TieredPGO>true</TieredPGO>` to your project file). PGO can also instrument and optimize additional things, such as delegates.
- Improved code generation for Arm64.
- Native AOT produces a standalone executable in the target platform's file format with no external dependencies. It's entirely native, with no IL or JIT, and provides fast startup time and a small, self-contained deployment. In .NET 7, Native AOT focuses on console apps and requires apps to be trimmed.
- Performance improvements to the Mono runtime, which powers Blazor WebAssembly, Android, and iOS apps.

For a detailed look at many of the performance-focused features that make .NET 7 so fast, see the Performance improvements in .NET 7 blog post.

System.Text.Json serialization

.NET 7 includes improvements to System.Text.Json serialization in the following areas:

- **Contract customization** gives you more control over how types are serialized and deserialized. For more information, see Customize a JSON contract.
- **Polymorphic serialization** for user-defined type hierarchies. For more information, see Serialize properties of derived classes.
- Support for **required members**, which are properties that must be present in the JSON payload for deserialization to succeed. For more information, see Required properties.

For information about these and other updates, see the What's new in System.Text.Json in .NET 7 blog post.

Generic math

.NET 7 and C# 11 include innovations that allow you to perform mathematical operations generically—that is, without having to know the exact type you're working with. For example, if you wanted to write a method that adds two numbers, previously you had to add an overload of the method for each type. Now you can write a single, generic method, where the type parameter is constrained to be a number-like type. For more information, see the Generic math article and the Generic math blog post.

Regular expressions

.NET's regular expression library has seen significant functional and performance improvements in .NET 7:

- The new option RegexOptions.NonBacktracking enables matching using an approach that avoids backtracking and guarantees linear-time processing in the length of the input. The nonbacktracking engine can't be used in a right-to-left search and has a few other restrictions, but is fast for all regular expressions and inputs. For more information, see Nonbacktracking mode.

- Regular expression source generators are new. Source generators build an engine that's optimized for *your* pattern at compile time, providing throughput performance benefits. The source that's emitted is part of your project, so you can view and debug it. In addition, a new source-generator diagnostic `SYSLIB1045` alerts you to places you use Regex that could be converted to the source generator. For more information, see .NET regular expression source generators.

- For case-insensitive searches, .NET 7 includes large performance gains. The gains come because specifying RegexOptions.IgnoreCase no longer calls ToLower on each character in the pattern and on each character in the input. Instead, all casing-related work is done when the Regex is constructed.

- Regex now supports spans for some APIs. The following new methods have been added as part of this support:
  - Regex.EnumerateMatches
  - Regex.Count
  - Regex.IsMatch(ReadOnlySpan<Char>) (and a few other overloads)

For more information about these and other improvements, see the Regular expression improvements in .NET 7 blog post.

.NET libraries

Many improvements have been made to .NET library APIs. Some are mentioned in other, dedicated sections of this article. Some others are summarized in the following table.

Expand table

Description	APIs	Further information
Support for microseconds and nanoseconds in TimeSpan, TimeOnly, DateTime, and DateTimeOffset types	- DateTime.Microsecond - DateTime.Nanosecond - DateTime.AddMicroseconds(Double) - New DateTime constructor overloads - DateTimeOffset.Microsecond - DateTimeOffset.Nanosecond - DateTimeOffset.AddMicroseconds(Double) - New DateTimeOffset constructor overloads - TimeOnly.Microsecond - TimeOnly.Nanosecond - TimeSpan.Microseconds - TimeSpan.Nanoseconds - TimeSpan.FromMicroseconds(Double) - And others...	These APIs mean you no longer have to perform computations on the "tick" value to determine microsecond and nanosecond values. For more information, see the .NET 7 Preview 4 blog post.
APIs for reading, writing, archiving, and extracting Tar archives	System.Formats.Tar	For more information, see the .NET 7 Preview 4 and .NET 7 Preview 6 blog posts.
Rate limiting APIs to protect a resource by	RateLimiter and others in the System.Threading.RateLimiting NuGet package	For more information, see Rate limit an HTTP handler in .NET and Announcing rate limiting for .NET.

Description	APIs	Further information
keeping traffic at a safe level		
APIs to read *all* the data from a Stream	- Stream.ReadExactly - Stream.ReadAtLeast	Stream.Read may return less data than what's available in the stream. The new `ReadExactly` methods read *exactly* the number of bytes requested, and the new `ReadAtLeast` methods read *at least* the number of bytes requested. For more information, see the .NET 7 Preview 5 blog post.
New type converters for `DateOnly`, `TimeOnly`, `Int128`, `UInt128`, and `Half`	In the System.ComponentModel namespace: - DateOnlyConverter - TimeOnlyConverter - Int128Converter - UInt128Converter - HalfConverter	Type converters are often used to convert value types to and from a string. These new APIs add type converters for types that were added more recently.
Metrics support for IMemoryCache	- MemoryCacheStatistics - MemoryCache.GetCurrentStatistics()	GetCurrentStatistics() lets you use event counters or metrics APIs to track statistics for one or more memory caches. For more information, see the .NET 7 Preview 4 blog post.
APIs to get and set Unix file permissions	- System.IO.UnixFileMode enum - File.GetUnixFileMode - File.SetUnixFileMode - FileSystemInfo.UnixFileMode - Directory.CreateDirectory(String, UnixFileMode) - FileStreamOptions.UnixCreateMode	For more information, see the .NET 7 Preview 7 blog post.
Attribute to indicate what kind of syntax is expected in a string	StringSyntaxAttribute	For example, you can specify that a `string` parameter expects a regular expression by attributing the parameter with `[StringSyntax(StringSyntaxAttribute.Regex)]`.
APIs to interop with JavaScript when running in the browser or other WebAssembly architectures	System.Runtime.InteropServices.JavaScript	JavaScript apps can use the expanded WebAssembly support in .NET 7 to reuse .NET libraries from JavaScript. For more information, see Use .NET from any JavaScript app in .NET 7.

Observability

.NET 7 makes improvements to *observability*. Observability helps you understand the state of your app as it scales and as the technical complexity increases. .NET's observability implementation is primarily built around OpenTelemetry. Improvements include:

- The new Activity.CurrentChanged event, which you can use to detect when the span context of a managed thread changes.
- New, performant enumerator methods for Activity properties: EnumerateTagObjects(), EnumerateLinks(), and EnumerateEvents().

For more information, see the .NET 7 Preview 4 blog post.

.NET SDK

The .NET 7 SDK improves the CLI template experience. It also enables publishing to containers, and central package management with NuGet.

Templates

Some welcome improvements have been made to the `dotnet new` command and to template authoring.

dotnet new

The dotnet new CLI command, which creates a new project, configuration file, or solution based on a template, now supports tab completion for exploring:

- Available template names
- Template options
- Allowable option values

In addition, for better conformity, the `install`, `uninstall`, `search`, `list`, and `update` subcommands no longer have the `--` prefix.

Authoring

Template *constraints*, a new concept for .NET 7, let you define the context in which your templates are allowed. Constraints help the template engine determine which templates it should show in commands like `dotnet new list`. You can constrain your template to an operating system, a template engine host (for example, the .NET CLI or New Project dialog in Visual Studio), and an installed workload. You define constraints in your template's configuration file.

Also in the template configuration file, you can now annotate a template parameter as allowing multiple values. For example, the web template allows multiple forms of authentication.

For more information, see the .NET 7 Preview 6 blog post.

Publish to a container

Containers are one of the easiest ways to distribute and run a wide variety of applications and services in the cloud. Container images are now a supported output type of the .NET SDK, and you can create containerized versions of your applications using dotnet publish. For more information about the feature, see Announcing built-in container support for the .NET SDK. For a tutorial, see Containerize a .NET app with dotnet publish.

Central package management

You can now manage common dependencies in your projects from one location using NuGet's central package management (CPM) feature. To enable it, you add a *Directory.Packages.props* file to the root of your repository. In this file, set the MSBuild property `ManagePackageVersionsCentrally` to `true` and add versions for common package dependency using `PackageVersion` items. Then, in the individual project files, you can omit `Version` attributes from any PackageReference items that refer to centrally managed packages.

For more information, see Central package management.

P/Invoke source generation

.NET 7 introduces a source generator for platform invokes (P/Invokes) in C#. The source generator looks for LibraryImportAttribute on `static`, `partial` methods to trigger compile-time source generation of marshalling code. By generating the marshalling code at compile time, no IL stub needs to be generated at run time, as it does when using DllImportAttribute. The source generator improves application performance and also allows the app to be ahead-of-time (AOT) compiled. For more information, see Source generation for platform invokes and Use custom marshallers in source-generated P/Invokes.

Related releases

This section contains information about related products that have releases that coincide with the .NET 7 release.

Visual Studio 2022 version 17.4

For more information, see What's new in Visual Studio 2022.

C# 11

C# 11 includes support for generic math, raw string literals, file-scoped types, and other new features. For more information, see What's new in C# 11.

F# 7

F# 7 continues the journey to make the language simpler and improve performance and interop with new C# features. For more information, see Announcing F# 7.

.NET MAUI

.NET Multi-platform App UI (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML. It unifies Android, iOS, macOS, and Windows APIs into a single API. For information about the latest updates, see What's new in .NET MAUI for .NET 7.

ASP.NET Core

ASP.NET Core 7.0 includes rate-limiting middleware, improvements to minimal APIs, and gRPC JSON transcoding. For information about all the updates, see What's new in ASP.NET Core 7.

EF Core

Entity Framework Core 7.0 includes provider-agnostic support for JSON columns, improved performance for saving changes, and custom reverse engineering templates. For information about all the updates, see What's new in EF Core 7.0.

Windows Forms

Much work has gone into Windows Forms for .NET 7. Improvements have been made in the following areas:

- Accessibility
- High DPI and scaling
- Databinding

For more information, see What's new in Windows Forms in .NET 7.

WPF

WPF in .NET 7 includes numerous bug fixes as well as performance and accessibility improvements. For more information, see the What's new for WPF in .NET 7 blog post.

Orleans

Orleans is a cross-platform framework for building robust, scalable distributed applications. For information about the latest updates for Orleans, see Migrate from Orleans 3.x to 7.0.

.NET Upgrade Assistant and CoreWCF

The .NET Upgrade Assistant now supports upgrading server-side WCF apps to CoreWCF, which is a community-created port of WCF to .NET (Core). For more information, see Upgrade a WCF server-side project to use CoreWCF.

ML.NET

ML.NET now includes a text classification API that makes it easy to train custom text classification models using the latest state-of-the-art deep learning techniques. For more information, see the What's new with AutoML and tooling and Introducing the ML.NET Text Classification API blog posts.

See also

- Release notes for .NET 7

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Breaking changes in .NET 7

Article • 07/28/2023

If you're migrating an app to .NET 7, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or Windows Forms.

This article indicates whether each breaking change is *binary compatible* or *source compatible*:

- **Binary compatible** - Existing binaries will load and execute successfully without recompilation, and the run-time behavior won't change.
- **Source compatible** - Source code will compile successfully without changes when targeting the new runtime or using the new SDK or component.

ASP.NET Core

Expand table

Title	Binary compatible	Source compatible
API controller actions try to infer parameters from DI	✔	✘
ASPNET-prefixed environment variable precedence	✔	✔
AuthenticateAsync for remote auth providers	✔	✘
Authentication in WebAssembly apps	✘	✔
Default authentication scheme	✘	✔
Event IDs for some Microsoft.AspNetCore.Mvc.Core log messages changed	✘	✔
Fallback file endpoints	✘	✔
IHubClients and IHubCallerClients hide members	✔	✘
Kestrel: Default HTTPS binding removed	✘	✔
Microsoft.AspNetCore.Server.Kestrel.Transport.Libuv and libuv.dll removed	✘	✘
Microsoft.Data.SqlClient updated to 4.0.1	✔	✘
Middleware no longer defers to endpoint with null request	✘	✔

Title	Binary compatible	Source compatible
delegate		
MVC's detection of an empty body in model binding changed	✗	✓
Output caching API changes	✗	✗
SignalR Hub methods try to resolve parameters from DI	✓	✗

Core .NET libraries

Expand table

Title	Binary compatible	Source compatible
API obsoletions with default diagnostic ID	✓	✗
API obsoletions with non-default diagnostic IDs	✓	✗
BinaryFormatter serialization APIs produce compiler errors	✓	✗
BrotliStream no longer allows undefined CompressionLevel values	✗	✓
C++/CLI projects in Visual Studio	✓	✗
Changes to reflection invoke API exceptions	✗	✓
Collectible Assembly in non-collectible AssemblyLoadContext	✗	✓
DateTime addition methods precision change	✓	✓
Equals method behavior change for NaN	✗	✓
EventSource callback behavior	✓	✓
Generic type constraint on PatternContext<T>	✗	✗
Legacy FileStream strategy removed	✗	✓
Library support for older frameworks	✗	✗
Maximum precision for numeric format strings	✗	✓
Regex patterns with ranges corrected	✓	✓
SerializationFormat.Binary is obsolete	✗	✗

Title	Binary compatible	Source compatible
System.Drawing.Common config switch removed	✔	✔
System.Runtime.CompilerServices.Unsafe NuGet package	✔	✔
Time fields on symbolic links	✘	✔
Tracking linked cache entries	✘	✔
Validate CompressionLevel for BrotliStream	✘	✔

Configuration

Expand table

Title	Binary compatible	Source compatible
System.diagnostics entry in app.config	✘	✔

Cryptography

Expand table

Title	Binary compatible	Source compatible
Decrypting EnvelopedCms doesn't double unwrap	✘	✔
Dynamic X509ChainPolicy verification time	✘	✔
X500DistinguishedName parsing of friendly names	✘	✔

Deployment

Expand table

Title	Binary compatible	Source compatible
All assemblies trimmed by default	✔	✘
Multi-level lookup is disabled	✘	✔
x86 host path on 64-bit Windows	✔	✔

Title	Binary compatible	Source compatible
TrimmerDefaultAction is deprecated	✔	✘

Entity Framework Core

Breaking changes in EF Core 7

Extensions

Expand table

Title	Binary compatible	Source compatible
Binding config to dictionary extends values	✔	✔
ContentRootPath for apps launched by Windows Shell	✘	✔
Environment variable prefixes	✘	✔

Globalization

Expand table

Title	Binary compatible	Source compatible
Globalization APIs use ICU libraries on Windows Server	✘	✔

Interop

Expand table

Title	Binary compatible	Source compatible
RuntimeInformation.OSArchitecture under emulation	✘	✔

.NET MAUI

Expand table

Title	Binary compatible	Source compatible
Constructors accept base interface instead of concrete type	✗	✓
Flow direction helper methods removed	✗	✗
New UpdateBackground parameter	✗	✓
ScrollToRequest property renamed	✗	✗
Some Windows APIs are removed	✗	✗

Networking

Expand table

Title	Binary compatible	Source compatible
AllowRenegotiation default is false	✗	✗
Custom ping payloads on Linux	✗	✓
Socket.End methods don't throw ObjectDisposedException	✗	✓

SDK and MSBuild

Expand table

Title	Binary compatible	Source compatible
Automatic RuntimeIdentifier for certain projects	✓	✗
Automatic RuntimeIdentifier for publish only	✗	✗
CLI console output uses UTF-8	✗	✗
Console encoding not UTF-8 after completion	✗	✓
MSBuild serialization of custom types in .NET 7	✗	✗

Title	Binary compatible	Source compatible
Side-by-side SDK installations	✗	✗
Tool manifests in root folder	✓	✓
Version requirements for .NET 7 SDK	✓	✓
dotnet test: switch -a to alias --arch instead of --test-adapter-path	✗	✗
dotnet test: switch -r to alias --runtime instead of --results-dir	✗	✗
--output option no longer is valid for solution-level commands	✗	✗
SDK no longer calls ResolvePackageDependencies	✓	✗

Serialization

Expand table

Title	Binary compatible	Source compatible
DataContractSerializer retains sign when deserializing -0	✗	✓
Deserialize Version type with leading or trailing whitespace	✗	✓
JsonSerializerOptions copy constructor includes JsonSerializerContext	✗	✓
Polymorphic serialization for object types	✗	✓
System.Text.Json source generator fallback	✗	✓

Windows Forms

Expand table

Title	Binary compatible	Source compatible
Obsoletions and warnings	✓	✗

Title	Binary compatible	Source compatible
Some APIs throw ArgumentNullException	✗	✓

XML and XSLT

Expand table

Title	Binary compatible	Source compatible
XmlSecureResolver is obsolete	✗	✗

See also

- What's new in .NET 7

What's new in .NET 6

Article • 05/26/2023

.NET 6 delivers the final parts of the .NET unification plan that started with .NET 5. .NET 6 unifies the SDK, base libraries, and runtime across mobile, desktop, IoT, and cloud apps. In addition to this unification, the .NET 6 ecosystem offers:

- **Simplified development**: Getting started is easy. New language features in C# 10 reduce the amount of code you need to write. And investments in the web stack and minimal APIs make it easy to quickly write smaller, faster microservices.
- **Better performance**: .NET 6 is the fastest full stack web framework, which lowers compute costs if you're running in the cloud.
- **Ultimate productivity**: .NET 6 and Visual Studio 2022 provide hot reload, new git tooling, intelligent code editing, robust diagnostics and testing tools, and better team collaboration.

.NET 6 will be supported for three years as a long-term support (LTS) release.

Preview features are disabled by default. They are also not supported for use in production and may be removed in a future version. The new RequiresPreviewFeaturesAttribute is used to annotate preview APIs, and a corresponding analyzer alerts you if you're using these preview APIs.

.NET 6 is supported by Visual Studio 2022 and Visual Studio 2022 for Mac (and later versions).

This article does not cover all of the new features of .NET 6. To see all of the new features, and for further information about the features listed in this article, see the Announcing .NET 6 blog post.

Performance

.NET 6 includes numerous performance improvements. This section lists some of the improvements—in FileStream, profile-guided optimization, and AOT compilation. For detailed information, see the Performance improvements in .NET 6 blog post.

FileStream

The System.IO.FileStream type has been rewritten for .NET 6 to provide better performance and reliability on Windows. Now, FileStream never blocks when created for

asynchronous I/O on Windows. For more information, see the File IO improvements in .NET 6 blog post.

Profile-guided optimization

Profile-guided optimization (PGO) is where the JIT compiler generates optimized code in terms of the types and code paths that are most frequently used. .NET 6 introduces *dynamic* PGO. Dynamic PGO works hand-in-hand with tiered compilation to further optimize code based on additional instrumentation that's put in place during tier 0. Dynamic PGO is disabled by default, but you can enable it with the `DOTNET_TieredPGO` environment variable. For more information, see JIT performance improvements.

Crossgen2

.NET 6 introduces Crossgen2, the successor to Crossgen, which has been removed. Crossgen and Crossgen2 are tools that provide ahead-of-time (AOT) compilation to improve the startup time of an app. Crossgen2 is written in C# instead of C++, and can perform analysis and optimization that weren't possible with the previous version. For more information, see Conversation about Crossgen2.

Arm64 support

The .NET 6 release includes support for macOS Arm64 (or "Apple Silicon") and Windows Arm64 operating systems, for both native Arm64 execution and x64 emulation. In addition, the x64 and Arm64 .NET installers now install side by side. For more information, see .NET Support for macOS 11 and Windows 11 for Arm64 and x64.

Hot reload

Hot reload is a feature that lets you modify your app's source code and instantly apply those changes to your running app. The feature's purpose is to increase your productivity by avoiding app restarts between edits. Hot reload is available in Visual Studio 2022 and the `dotnet watch` command-line tool. Hot reload works with most types of .NET apps, and for C#, Visual Basic, and C++ source code. For more information, see the Hot reload blog post.

.NET MAUI

.NET Multi-platform App UI (.NET MAUI) is still in *preview*, with a release candidate coming in the first quarter of 2022 and general availability (GA) in the second quarter of 2022. .NET MAUI makes it possible to build native client apps for desktop and mobile operating systems with a single codebase. For more information, see the Update on .NET Multi-platform App UI blog post.

C# 10 and templates

C# 10 includes innovations such as `global using` directives, file-scoped namespace declarations, and record structs. For more information, see What's new in C# 10.

In concert with that work, the .NET SDK project templates for C# have been modernized to use some of the new language features:

- `async Main` method
- Top-level statements
- Target-typed new expressions
- Implicit global using directives
- File-scoped namespaces
- Nullable reference types

By adding these new language features to the project templates, new code starts with the features enabled. However, existing code isn't affected when you upgrade to .NET 6. For more information about these template changes, see the .NET SDK: C# project templates modernized blog post.

F# and Visual Basic

F# 6 adds several improvements to the F# language and F# Interactive. For more information, see What's new in F# 6.

Visual Basic has improvements in the Visual Studio experience and Windows Forms project startup.

SDK Workloads

To keep the size of the .NET SDK smaller, some components have been placed in new, optional *SDK workloads*. These components include .NET MAUI and Blazor WebAssembly AOT. If you use Visual Studio, it will take care of installing any SDK workloads that you need. If you use the .NET CLI, you can manage workloads using the new `dotnet workload` commands:

Command	Description
dotnet workload search	Searches for available workloads.
dotnet workload install	Installs a specified workload.
dotnet workload uninstall	Removes a specified workload.
dotnet workload update	Updates installed workloads.
dotnet workload repair	Reinstalls all installed workloads to repair a broken installation.
dotnet workload list	Lists installed workloads.

For more information, see Optional SDK workloads.

System.Text.Json APIs

Many improvements have been made in System.Text.Json in .NET 6, such that it is now an "industrial strength" serialization solution.

Source generator

.NET 6 adds a new source generator for System.Text.Json. Source generation works with JsonSerializer and can be configured in multiple ways. It can improve performance, reduce memory usage, and facilitate assembly trimming. For more information, see How to choose reflection or source generation in System.Text.Json and How to use source generation in System.Text.Json.

Writeable DOM

A new, writeable document object model (DOM) has been added, which supplements the pre-existing read-only DOM. The new API provides a lightweight serialization alternative for cases when use of plain old CLR object (POCO) types isn't possible. It also allows you to efficiently navigate to a subsection of a large JSON tree and read an array or deserialize a POCO from that subsection. The following new types have been added to support the writeable DOM:

- JsonNode
- JsonArray
- JsonObject
- JsonValue

For more information, see JSON DOM choices.

IAsyncEnumerable serialization

System.Text.Json now supports serialization and deserialization with IAsyncEnumerable<T> instances. Asynchronous serialization methods enumerate any IAsyncEnumerable<T> instances in an object graph and then serialize them as JSON arrays. For deserialization, the new method JsonSerializer.DeserializeAsyncEnumerable<TValue>(Stream, JsonSerializerOptions, CancellationToken) was added. For more information, see IAsyncEnumerable serialization.

Other new APIs

New serialization interfaces for validation and defaulting values:

- IJsonOnDeserialized
- IJsonOnDeserializing
- IJsonOnSerialized
- IJsonOnSerializing

For more information, see Callbacks.

New property ordering attribute:

- JsonPropertyOrderAttribute

  For more information, see Configure the order of serialized properties.

New method to write "raw" JSON:

- Utf8JsonWriter.WriteRawValue

  For more information, see Write Raw JSON.

Synchronous serialization and deserialization to a stream:

- JsonSerializer.Deserialize(Stream, Type, JsonSerializerOptions)
- JsonSerializer.Deserialize(Stream, Type, JsonSerializerContext)
- JsonSerializer.Deserialize<TValue>(Stream, JsonSerializerOptions)
- JsonSerializer.Deserialize<TValue>(Stream, JsonTypeInfo<TValue>)
- JsonSerializer.Serialize(Stream, Object, Type, JsonSerializerOptions)
- JsonSerializer.Serialize(Stream, Object, Type, JsonSerializerContext)
- JsonSerializer.Serialize<TValue>(Stream, TValue, JsonSerializerOptions)
- JsonSerializer.Serialize<TValue>(Stream, TValue, JsonTypeInfo<TValue>)

New option to ignore an object when a reference cycle is detected during serialization:

- ReferenceHandler.IgnoreCycles

  For more information, see Ignore circular references.

For more information about serializing and deserializing with `System.Text.Json`, see JSON serialization and deserialization in .NET.

HTTP/3

.NET 6 includes preview support for HTTP/3, a new version of HTTP. HTTP/3 solves some existing functional and performance challenges by using a new underlying connection protocol called QUIC. QUIC establishes connections more quickly, and connections are independent of the IP address, allowing mobile clients to roam between Wi-fi and cellular networks. For more information, see Use HTTP/3 with HttpClient.

ASP.NET Core

ASP.NET Core includes improvements in minimal APIs, ahead-of-time (AOT) compilation for Blazor WebAssembly apps, and single-page apps. In addition, Blazor components can now be rendered from JavaScript and integrated with existing JavaScript based apps. For more information, see What's new in ASP.NET Core 6.

OpenTelemetry

.NET 6 brings improved support for OpenTelemetry, which is a collection of tools, APIs, and SDKs that help you analyze your software's performance and behavior. APIs in the System.Diagnostics.Metrics namespace implement the OpenTelemetry Metrics API specification. For example, there are four instrument classes to support different metrics scenarios. The instrument classes are:

- Counter<T>
- Histogram<T>
- ObservableCounter<T>
- ObservableGauge<T>

Security

.NET 6 adds preview support for two key security mitigations: Control-flow Enforcement Technology (CET) and "write exclusive execute" (W^X).

CET is an Intel technology available in some newer Intel and AMD processors. It adds capabilities to the hardware that protect against some control-flow hijacking attacks. .NET 6 provides support for CET for Windows x64 apps, and you must explicitly enable it. For more information, see .NET 6 compatibility with Intel CET shadow stacks.

W^X is available all operating systems with .NET 6 but only enabled by default on Apple Silicon. W^X blocks the simplest attack path by disallowing memory pages to be writeable and executable at the same time.

IL trimming

Trimming of self-contained deployments is improved. In .NET 5, only unused assemblies were trimmed. .NET 6 adds trimming of unused types and members too. In addition, trim warnings, which alert you to places where trimming may remove code that's used at run time, are now *enabled* by default. For more information, see Trim self-contained deployments and executables.

Code analysis

The .NET 6 SDK includes a handful of new code analyzers that concern API compatibility, platform compatibility, trimming safety, use of span in string concatenation and splitting, faster string APIs, and faster collection APIs. For a full list of new (and removed) analyzers, see Analyzer releases - .NET 6.

Custom platform guards

The Platform compatibility analyzer recognizes the `Is<Platform>` methods in the OperatingSystem class, for example, OperatingSystem.IsWindows(), as platform guards. To allow for custom platform guards, .NET 6 introduces two new attributes that you can use to annotate fields, properties, or methods with a supported or unsupported platform name:

- SupportedOSPlatformGuardAttribute
- UnsupportedOSPlatformGuardAttribute

Windows Forms

Application.SetDefaultFont(Font) is a new method in .NET 6 that sets the default font across your application.

The templates for C# Windows Forms apps have been updated to support `global using` directives, file-scoped namespaces, and nullable reference types. In addition, they include application bootstrap code, which reduces boilerplate code and allows the Windows Forms designer to render the design surface in the preferred font. The bootstrap code is a call to `ApplicationConfiguration.Initialize()`, which is a source-generated method that emits calls to other configuration methods, such as Application.EnableVisualStyles(). Additionally, if you set a non-default font via the ApplicationDefaultFont MSBuild property, `ApplicationConfiguration.Initialize()` emits a call to SetDefaultFont(Font).

For more information, see the What's new in Windows Forms blog post.

Source build

The *source tarball*, which contains all the source for the .NET SDK, is now a product of the .NET SDK build. Other organizations, such as Red Hat, can build their own version of the SDK using this source tarball.

Target framework monikers

Additional OS-specific target framework monikers (TFMs) have been added for .NET 6, for example, `net6.0-android`, `net6.0-ios`, and `net6.0-macos`. For more information, see .NET 5+ OS-specific TFMs.

Generic math

In *preview* is the ability to use operators on generic types in .NET 6. .NET 6 introduces numerous interfaces that make use of C# 10's new preview feature, `static abstract` interface members. These interfaces correspond to different operators, for example, `IAdditionOperators` represents the `+` operator. The interfaces are available in the System.Runtime.Experimental NuGet package. For more information, see the Generic math blog post.

NuGet package validation

If you're a NuGet library developer, new package-validation tooling enables you to validate that your packages are consistent and well-formed. You can determine if:

- There are any breaking changes across package versions.

- The package has the same set of public APIs for all runtime-specific implementations.
- There are any gaps for target-framework or runtime applicability.

For more information, see the Package Validation blog post.

Reflection APIs

.NET 6 introduces the following new APIs that inspect code and provide nullability information:

- System.Reflection.NullabilityInfo
- System.Reflection.NullabilityInfoContext
- System.Reflection.NullabilityState

These APIs are useful for reflection-based tools and serializers.

Microsoft.Extensions APIs

Several extensions namespaces have improvements in .NET 6, as the following table shows.

Namespace	Improvements
Microsoft.Extensions.DependencyInjection	CreateAsyncScope lets you safely use a `using` statement for a service provider that registers an IAsyncDisposable service.
Microsoft.Extensions.Hosting	New ConfigureHostOptions methods simplify application setup.
Microsoft.Extensions.Logging	Microsoft.Extensions.Logging has a new source generator for performant logging APIs. The source generator is triggered if you add the new LoggerMessageAttribute to a `partial` logging method. At compile time, the generator generates the implementation of the `partial` method, which is typically faster at run time than existing logging solutions. For more information, see Compile-time logging source generation.

New LINQ APIs

Numerous LINQ methods have been added in .NET 6. Most of the new methods listed in the following table have equivalent methods in the System.Linq.Queryable type.

Method	Description
Enumerable.TryGetNonEnumeratedCount<TSource>(IEnumerable<TSource>, Int32)	Attempts to determine the number of elements in a sequence without forcing an enumeration.
Enumerable.Chunk<TSource>(IEnumerable<TSource>, Int32)	Splits the elements of a sequence into chunks of a specified size.
Enumerable.MaxBy and Enumerable.MinBy	Finds maximal or minimal elements using a key selector.
Enumerable.DistinctBy, Enumerable.ExceptBy, Enumerable.IntersectBy, and Enumerable.UnionBy	These new variations of methods that perform set-based operations let you specify equality using a key selector function.
Enumerable.ElementAt<TSource>(IEnumerable<TSource>, Index) and Enumerable.ElementAtOrDefault<TSource>(IEnumerable<TSource>, Index)	Accepts indexes counted from the beginning or end of the sequence—for example, `Enumerable.Range(1, 10).ElementAt(^2)` returns `9`.
Enumerable.FirstOrDefault<TSource>(IEnumerable<TSource>, TSource) and Enumerable.FirstOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource) Enumerable.LastOrDefault<TSource>(IEnumerable<TSource>, TSource) and Enumerable.LastOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource) Enumerable.SingleOrDefault<TSource>(IEnumerable<TSource>, TSource) and Enumerable.SingleOrDefault<TSource>(IEnumerable<TSource>, Func<TSource,Boolean>, TSource)	New overloads let you specify a default value to use if the sequence is empty.
Enumerable.Max<TSource>(IEnumerable<TSource>, IComparer<TSource>) and Enumerable.Min<TSource>(IEnumerable<TSource>, IComparer<TSource>)	New overloads let you specify a comparer.
Enumerable.Take<TSource>(IEnumerable<TSource>, Range)	Accepts a Range argument to simplify taking a slice of a sequence—for example, you can use `source.Take(2..7)` instead of `source.Take(7).Skip(2)`.

Method	Description
Enumerable.Zip<TFirst,TSecond,TThird> (IEnumerable<TFirst>, IEnumerable<TSecond>, IEnumerable<TThird>)	Produces a sequence of tuples with elements from *three* specified sequences.

Date, time, and time zone improvements

The following two structs were added in .NET 6: System.DateOnly and System.TimeOnly. These represent the date part and the time part of a DateTime, respectively. DateOnly is useful for birthdays and anniversaries, and TimeOnly is useful for daily alarms and weekly business hours.

You can now use either Internet Assigned Numbers Authority (IANA) or Windows time zone IDs on any operating system that has time zone data installed. The TimeZoneInfo.FindSystemTimeZoneById(String) method has been updated to automatically convert its input from a Windows time zone to an IANA time zone (or vice versa) if the requested time zone is not found on the system. In addition, the new methods TryConvertIanaIdToWindowsId(String, String) and TryConvertWindowsIdToIanaId have been added for scenarios when you still need to manually convert from one time zone format to another.

There are a few other time zone improvements as well. For more information, see Date, Time, and Time Zone Enhancements in .NET 6.

PriorityQueue class

The new PriorityQueue<TElement,TPriority> class represents a collection of items that have both a value and a priority. Items are dequeued in increasing priority order—that is, the item with the lowest priority value is dequeued first. This class implements a min heap data structure.

See also

- What's new in C# 10
- What's new in F# 6
- What's new in EF Core 6
- What's new in ASP.NET Core 6
- Release notes for .NET 6
- Release notes for Visual Studio 2022
- Blog: Announcing .NET 6

- Blog: Try the new System.Text.Json source generator

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Breaking changes in .NET 6

Article • 11/21/2023

If you're migrating an app to .NET 6, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or Windows Forms.

This article indicates whether each breaking change is *binary compatible* or *source compatible*:

- **Binary compatible** - Existing binaries will load and execute successfully without recompilation, and the run-time behavior won't change.
- **Source compatible** - Source code will compile successfully without changes when targeting the new runtime or using the new SDK or component.

ASP.NET Core

Expand table

Title	Binary compatible	Source compatible
ActionResult<T> sets StatusCode to 200	✔	✘
AddDataAnnotationsValidation method made obsolete	✔	✘
Assemblies removed from Microsoft.AspNetCore.App shared framework	✘	✔
Blazor: Parameter name changed in RequestImageFileAsync method	✔	✘
Blazor: WebEventDescriptor.EventArgsType property replaced	✘	✘
Blazor: Byte array interop	✔	✘
Changed MessagePack library in @microsoft/signalr-protocol-msgpack	✘	✔
ClientCertificate property doesn't trigger renegotiation for HttpSys	✔	✘
EndpointName metadata not set automatically	✔	✘
Identity: Default Bootstrap version of UI changed	✘	✘
Kestrel: Log message attributes changed	✔	✘

Title	Binary compatible	Source compatible
Microsoft.AspNetCore.Http.Features split	✗	✓
Middleware: HTTPS Redirection Middleware throws exception on ambiguous HTTPS ports	✓	✗
Middleware: New Use overload	✓	✗
Minimal API renames in RC 1	✗	✗
Minimal API renames in RC 2	✗	✗
MVC doesn't buffer IAsyncEnumerable types when using System.Text.Json	✓	✗
Nullable reference type annotations changed	✓	✗
Obsoleted and removed APIs	✓	✗
PreserveCompilationContext not configured by default	✗	✓
Razor: Compiler no longer produces a Views assembly	✓	✗
Razor: Logging ID changes	✗	✓
Razor: RazorEngine APIs marked obsolete	✓	✗
SignalR: Java Client updated to RxJava3	✗	✓
TryParse and BindAsync methods are validated	✗	✗

Containers

Expand table

Title	Binary compatible	Source compatible
Default console logger formatting in container images	✓	✗

For information on other breaking changes for containers in .NET 6, see .NET 6 Container Release Notes.

Core .NET libraries

Expand table

Title	Binary compatible	Source compatible
API obsoletions with non-default diagnostic IDs	✔	✘
Changes to nullable reference type annotations	✔	✘
Conditional string evaluation in Debug methods	✔	✘
Environment.ProcessorCount behavior on Windows	✔	✘
EventSource callback behavior	✔	✔
File.Replace on Unix throws exceptions to match Windows	✔	✘
FileStream locks files with shared lock on Unix	✘	✔
FileStream no longer synchronizes file offset with OS	✘	✘
FileStream.Position updates after ReadAsync or WriteAsync completes	✘	✘
New diagnostic IDs for obsoleted APIs	✔	✘
New System.Linq.Queryable method overloads	✔	✘
Older framework versions dropped from package	✘	✔
Parameter names changed	✔	✘
Parameter names in Stream-derived types	✔	✘
Partial and zero-byte reads in DeflateStream, GZipStream, and CryptoStream	✔	✘
Set timestamp on read-only file on Windows	✘	✔
Standard numeric format parsing precision	✔	✘
Static abstract members in interfaces	✘	✔
StringBuilder.Append overloads and evaluation order	✘	✔
Strong-name APIs throw PlatformNotSupportedException	✘	✔
System.Drawing.Common only supported on Windows	✘	✘
System.Security.SecurityContext is marked obsolete	✔	✘
Task.FromResult may return singleton	✘	✔

Title	Binary compatible	Source compatible
Unhandled exceptions from a BackgroundService	✔	✘

Cryptography

Expand table

Title	Binary compatible	Source compatible
CreateEncryptor methods throw exception for incorrect feedback size	✘	✔

Deployment

Expand table

Title	Binary compatible	Source compatible
x86 host path on 64-bit Windows	✔	✔

Entity Framework Core

Breaking changes in EF Core 6

Extensions

Expand table

Title	Binary compatible	Source compatible
AddProvider checks for non-null provider	✔	✘
FileConfigurationProvider.Load throws InvalidDataException	✔	✘
Repeated XML elements include index	✘	✔
Resolving disposed ServiceProvider throws exception	✔	✘

Globalization

Expand table

Title	Binary compatible	Source compatible
Culture creation and case mapping in globalization-invariant mode		

Interop

Expand table

Title	Binary compatible	Source compatible
Static abstract members in interfaces	✗	✓

JIT compiler

Expand table

Title	Binary compatible	Source compatible
Coerce call arguments according to ECMA-335	✓	✓

Networking

Expand table

Title	Binary compatible	Source compatible
Port removed from SPN for Kerberos and Negotiate	✗	✓
WebRequest, WebClient, and ServicePoint are obsolete	✓	✗

SDK

Expand table

Title	Binary compatible	Source compatible
-p option for dotnet run is deprecated	✔	✘
C# code in templates not supported by earlier versions	✔	✔
EditorConfig files implicitly included	✔	✘
Generate apphost for macOS	✔	✘
Generate error for duplicate files in publish output	✘	✔
GetTargetFrameworkProperties and GetNearestTargetFramework removed from ProjectReference protocol	✘	✔
Install location for x64 emulated on Arm64	✔	✘
MSBuild no longer supports calling GetType()		
.NET can't be installed to custom location	✔	✔
OutputType not automatically set to WinExe	✔	✘
Publish ReadyToRun with --no-restore requires changes	✔	✘
runtimeconfig.dev.json file not generated	✘	✔
RuntimeIdentifier warning if self-contained is unspecified	✔	✘
Tool manifests in root folder	✔	✔
Version requirements for .NET 6 SDK	✔	✔
.version file includes build version	✔	✔
Write reference assemblies to IntermediateOutputPath	✘	✔

Serialization

Expand table

Title	Binary compatible	Source compatible
DataContractSerializer retains sign when deserializing -0	✘	✔

Title	Binary compatible	Source compatible
Default serialization format for TimeSpan	✗	✓
IAsyncEnumerable serialization	✓	✗
JSON source-generation API refactoring	✗	✓
JsonNumberHandlingAttribute on collection properties	✗	✓
New JsonSerializer source generator overloads	✗	✓

Windows Forms

Expand table

Title	Binary compatible	Source compatible
C# templates use application bootstrap	✓	✗
Selected TableLayoutSettings properties throw InvalidEnumArgumentException	✗	✓
DataGridView-related APIs now throw InvalidOperationException	✗	✓
ListViewGroupCollection methods throw new InvalidOperationException	✗	✓
NotifyIcon.Text maximum text length increased	✗	✓
ScaleControl called only when needed	✓	✗
Some APIs throw ArgumentNullException	✗	✓
TreeNodeCollection.Item throws exception if node is assigned elsewhere	✗	✓

XML and XSLT

Expand table

Title	Binary compatible	Source compatible
XNodeReader.GetAttribute behavior for invalid index	✔	✘

See also

- What's new in .NET 6

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

What's new in .NET 5

Article • 09/22/2022

.NET 5 is the next major release of .NET Core following 3.1. We named this new release .NET 5 instead of .NET Core 4 for two reasons:

- We skipped version numbers 4.x to avoid confusion with .NET Framework 4.x.
- We dropped "Core" from the name to emphasize that this is the main implementation of .NET going forward. .NET 5 supports more types of apps and more platforms than .NET Core or .NET Framework.

ASP.NET Core 5.0 is based on .NET 5 but retains the name "Core" to avoid confusing it with ASP.NET MVC 5. Likewise, Entity Framework Core 5.0 retains the name "Core" to avoid confusing it with Entity Framework 5 and 6.

.NET 5 includes the following improvements and new features compared to .NET Core 3.1:

- C# updates
- F# updates
- Visual Basic updates
- System.Text.Json new features
- Single file apps
- App trimming
- Windows Arm64 and Arm64 intrinsics
- Tooling support for dump debugging
- The runtime libraries are 80% annotated for nullable reference types
- Performance improvements:
  - Garbage Collection (GC)
  - System.Text.Json
  - System.Text.RegularExpressions
  - Async ValueTask pooling
  - Container size optimizations
  - Many more areas

.NET 5 doesn't replace .NET Framework

.NET 5 and later versions are the main implementation of .NET going forward, but .NET Framework 4.x is still supported. There are no plans to port the following technologies from .NET Framework to .NET 5, but there are alternatives in .NET:

Technology	Recommended alternative
Web Forms	ASP.NET Core Blazor or Razor Pages
Windows Workflow (WF)	Elsa-Workflows

Windows Communication Foundation

The original implementation of Windows Communication Foundation (WCF) was only supported on Windows. However, there's a client port available from the .NET Foundation. It's entirely open source, cross platform, and supported by Microsoft. The core NuGet packages are listed below:

- System.ServiceModel.Duplex
- System.ServiceModel.Federation
- System.ServiceModel.Http
- System.ServiceModel.NetTcp
- System.ServiceModel.Primitives
- System.ServiceModel.Security

The server components that complement the aforementioned client libraries are available through CoreWCF. As of April 2022, CoreWCF is officially supported by Microsoft. However, for an alternative to WCF, consider gRPC.

.NET 5 doesn't replace .NET Standard

New application development can specify the `net5.0` Target Framework Moniker (TFM) for all project types, including class libraries. Sharing code between .NET 5 workloads is simplified: all you need is the `net5.0` TFM.

For .NET 5 apps and libraries, the `net5.0` TFM combines and replaces the `netcoreapp` and `netstandard` TFMs. However, if you plan to share code between .NET Framework, .NET Core, and .NET 5 workloads, you can do so by specifying `netstandard2.0` as your TFM. For more information, see .NET Standard.

C# updates

Developers writing .NET 5 apps will have access to the latest C# version and features. .NET 5 is paired with C# 9, which brings many new features to the language. Here are a few highlights:

- **Records**: Reference types with value-based equality semantics and non-destructive mutation supported by a new `with` expression.
- **Relational pattern matching**: Extends pattern matching capabilities to relational operators for comparative evaluations and expressions, including logical patterns - new keywords `and`, `or`, and `not`.
- **Top-level statements**: As a means for accelerating the adoption and learning of C#, the `Main` method can be omitted, and an application as simple as the following example is valid:

  ```
  C#

  System.Console.Write("Hello world!");
  ```

- **Function pointers**: Language constructs that expose the following intermediate language (IL) opcodes: `ldftn` and `calli`.

For more information on the available C# 9 features, see What's new in C# 9.

Source generators

In addition to some of the highlighted new C# features, source generators are making their way into developer projects. Source generators allow code that runs during compilation to inspect your program and produce additional files that are compiled together with the rest of your code.

For more information on source generators, see Introducing C# source generators and C# source generator samples.

F# updates

F# is the .NET functional programming language, and with .NET 5, developers have access to F# 5. One of the new features is interpolated strings, similar to interpolated strings in C#, and even JavaScript.

```
F#

let name = "David"
let age = 36
let message = $"{name} is {age} years old."
```

In addition to basic string interpolation, there's typed interpolation. With typed interpolation, a given type must match the format specifier.

```F#
let name = "David"
let age = 36
let message = $"%s{name} is %d{age} years old."
```

This format is similar to the sprintf function that formats a string based on type-safe inputs.

For more information, see What's new in F# 5.

Visual Basic updates

There are no new language features for Visual Basic in .NET 5. However, with .NET 5, Visual Basic support is extended to:

Description	`dotnet new` **parameter**
Console Application	`console`
Class library	`classlib`
WPF Application	`wpf`
WPF Class library	`wpflib`
WPF Custom Control Library	`wpfcustomcontrollib`
WPF User Control Library	`wpfusercontrollib`
Windows Forms (WinForms) Application	`winforms`
Windows Forms (WinForms) Class library	`winformslib`
Unit Test Project	`mstest`
NUnit 3 Test Project	`nunit`
NUnit 3 Test Item	`nunit-test`
xUnit Test Project	`xunit`

For more information on project templates from the .NET CLI, see dotnet new.

System.Text.Json new features

There are new features in and for System.Text.Json:

- Preserve references and handle circular references
- HttpClient and HttpContent extension methods
- Allow or write numbers in quotes
- Support immutable types and C# 9 Records
- Support non-public property accessors
- Support fields
- Conditionally ignore properties
- Support non-string-key dictionaries
- Allow custom converters to handle null
- Copy JsonSerializerOptions
- Create JsonSerializerOptions with web defaults

See also

- The Journey to one .NET
- Performance improvements in .NET 5
- Download the .NET SDK

Breaking changes in .NET 5

Article • 03/18/2023

If you're migrating an app to .NET 5, the breaking changes listed here might affect you. Changes are grouped by technology area, such as ASP.NET Core or cryptography.

This article indicates whether each breaking change is *binary compatible* or *source compatible*:

- **Binary compatible** - Existing binaries will load and execute successfully without recompilation, and the run-time behavior won't change.
- **Source compatible** - Source code will compile successfully without changes when targeting the new runtime or using the new SDK or component.

ASP.NET Core

Title	Binary compatible	Source compatible
ASP.NET Core apps deserialize quoted numbers	✔	✘
AzureAD.UI and AzureADB2C.UI APIs obsolete	✔	✘
BinaryFormatter serialization methods are obsolete	✔	✘
Resource in endpoint routing is HttpContext	✔	✘
Microsoft-prefixed Azure integration packages removed	✘	✔
Blazor: Route precedence logic changed in Blazor apps	✔	✘
Blazor: Updated browser support	✔	✔
Blazor: Insignificant whitespace trimmed by compiler	✔	✘
Blazor: JSObjectReference and JSInProcessObjectReference types are internal	✔	✘
Blazor: Target framework of NuGet packages changed	✘	✔
Blazor: ProtectedBrowserStorage feature moved to shared framework	✔	✘
Blazor: RenderTreeFrame readonly public fields are now properties	✘	✔
Blazor: Updated validation logic for static web assets	✘	✔

Title	Binary compatible	Source compatible
Cryptography APIs not supported on browser	✗	✓
Extensions: Package reference changes	✗	✓
Kestrel and IIS BadHttpRequestException types are obsolete	✓	✗
HttpClient instances created by IHttpClientFactory log integer status codes	✓	✗
HttpSys: Client certificate renegotiation disabled by default	✓	✗
IIS: UrlRewrite middleware query strings are preserved	✓	✗
Kestrel: Configuration changes detected by default	✓	✗
Kestrel: Default supported TLS protocol versions changed	✓	✗
Kestrel: HTTP/2 disabled over TLS on incompatible Windows versions	✓	✓
Kestrel: Libuv transport marked as obsolete	✓	✗
Obsolete properties on ConsoleLoggerOptions	✓	✗
ResourceManagerWithCultureStringLocalizer class and WithCulture interface member removed	✓	✗
Pubternal APIs removed	✓	✗
Obsolete constructor removed in request localization middleware	✓	✗
Middleware: Database error page marked as obsolete	✓	✗
Exception handler middleware throws original exception	✓	✓
ObjectModelValidator calls a new overload of Validate	✓	✗
Cookie name encoding removed	✓	✗
IdentityModel NuGet package versions updated	✗	✓
SignalR: MessagePack Hub Protocol options type changed	✓	✗
SignalR: MessagePack Hub Protocol moved	✓	✗
UseSignalR and UseConnections methods removed	✓	✗
CSV content type changed to standards-compliant	✓	✗

Code analysis

Title	Binary compatible	Source compatible
CA1416 warning	✔	✘
CA1417 warning	✔	✘
CA1831 warning	✔	✘
CA2013 warning	✔	✘
CA2014 warning	✔	✘
CA2015 warning	✔	✘
CA2200 warning	✔	✘
CA2247 warning	✔	✘

Core .NET libraries

Title	Binary compatible	Source compatible
Assembly-related API changes for single-file publishing	✘	✔
BinaryFormatter serialization methods are obsolete	✔	✘
Code access security APIs are obsolete	✔	✘
CreateCounterSetInstance throws InvalidOperationException	✔	✘
Default ActivityIdFormat is W3C	✘	✔
Environment.OSVersion returns the correct version	✘	✔
FrameworkDescription's value is .NET not .NET Core	✔	✘
GAC APIs are obsolete	✔	✘
Hardware intrinsic IsSupported checks	✘	✔
IntPtr and UIntPtr implement IFormattable	✔	✘
LastIndexOf handles empty search strings	✘	✔
URI paths with non-ASCII characters on Unix	✘	✔

Title	Binary compatible	Source compatible
API obsoletions with non-default diagnostic IDs	✔	✘
Obsolete properties on ConsoleLoggerOptions	✔	✘
Complexity of LINQ OrderBy.First	✘	✔
OSPlatform attributes renamed or removed	✔	✘
Microsoft.DotNet.PlatformAbstractions package removed	✘	✔
PrincipalPermissionAttribute is obsolete	✔	✘
Parameter name changes from preview versions	✔	✘
Parameter name changes in reference assemblies	✔	✘
Remoting APIs are obsolete	✘	✔
Order of Activity.Tags list is reversed	✔	✘
SSE and SSE2 comparison methods handle NaN	✔	✘
Thread.Abort is obsolete	✔	✘
Uri recognition of UNC paths on Unix	✘	✔
UTF-7 code paths are obsolete	✔	✘
Behavior change for Vector2.Lerp and Vector4.Lerp	✔	✘
Vector<T> throws NotSupportedException	✘	✔

Cryptography

Title	Binary compatible	Source compatible
Cryptography APIs not supported on browser	✘	✔
Cryptography.Oid is init-only	✔	✘
Default TLS cipher suites on Linux	✘	✔
Create() overloads on cryptographic abstractions are obsolete	✔	✘
Default FeedbackSize value changed	✔	✘

Entity Framework Core

Breaking changes in EF Core 5.0

Globalization

Title	Binary compatible	Source compatible
Use ICU libraries on Windows	✗	✓
StringInfo and TextElementEnumerator are UAX29-compliant	✗	✓
Unicode category changed for Latin-1 characters	✓	✗
TextInfo.ListSeparator values changed	✓	✗

Interop

Title	Binary compatible	Source compatible
Support for WinRT is removed	✗	✓
Casting RCW to InterfaceIsIInspectable throws exception	✗	✓
No A/W suffix probing on non-Windows platforms	✗	✓

Networking

Title	Binary compatible	Source compatible
Cookie path handling conforms to RFC 6265	✓	✗
LocalEndPoint is updated after calling SendToAsync	✓	✗
MulticastOption.Group doesn't accept null	✓	✗
Streams allow successive Begin operations	✗	✓
WinHttpHandler removed from .NET runtime	✗	✓

SDK

Title	Binary compatible	Source compatible
Directory.Packages.props files imported by default	✗	✓
Error generated when executable project references mismatched executable		✓
FrameworkReference replaced with WindowsSdkPackageVersion for Windows SDK	✓	✗
NETCOREAPP3_1 preprocessor symbol not defined	✓	✗
OutputType set to WinExe	✗	✓
PublishDepsFilePath behavior change	✗	✓
TargetFramework change from netcoreapp to net	✗	✓
WinForms and WPF apps use Microsoft.NET.Sdk	✗	✓

Security

Title	Binary compatible	Source compatible
Code access security APIs are obsolete	✓	✗
PrincipalPermissionAttribute is obsolete	✓	✗
UTF-7 code paths are obsolete	✓	✗

Serialization

Title	Binary compatible	Source compatible
BinaryFormatter.Deserialize rewraps exceptions	✓	✗
JsonSerializer.Deserialize requires single-character string	✓	✗
ASP.NET Core apps deserialize quoted numbers	✓	✗
JsonSerializer.Serialize throws ArgumentNullException	✓	✗
Non-public, parameterless constructors not used for	✓	✗

Title	Binary compatible	Source compatible
deserialization		
Options are honored when serializing key-value pairs	✔	✘

Windows Forms

Title	Binary compatible	Source compatible
Native code can't access Windows Forms objects	✔	✘
OutputType set to WinExe	✘	✔
DataGridView doesn't reset custom fonts	✔	✘
Methods throw ArgumentException	✔	✘
Methods throw ArgumentNullException	✔	✘
Properties throw ArgumentOutOfRangeException	✔	✘
TextFormatFlags.ModifyString is obsolete	✔	✘
DataGridView APIs throw InvalidOperationException	✔	✘
WinForms apps use Microsoft.NET.Sdk	✘	✔
Removed status bar controls	✔	✘

WPF

Title	Binary compatible	Source compatible
OutputType set to WinExe	✘	✔
WPF apps use Microsoft.NET.Sdk	✘	✔

See also

- What's new in .NET 5

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Core 3.1

Article • 07/12/2022

This article describes what is new in .NET Core 3.1. This release contains minor improvements to .NET Core 3.0, focusing on small, but important, fixes. The most important feature about .NET Core 3.1 is that it's a long-term support (LTS) release.

If you're using Visual Studio 2019, you must update to Visual Studio 2019 version 16.4 or later to work with .NET Core 3.1 projects. For information on what's new in Visual Studio version 16.4, see What's New in Visual Studio 2019 version 16.4.

Visual Studio for Mac also supports and includes .NET Core 3.1 in Visual Studio for Mac 8.4.

For more information about the release, see the .NET Core 3.1 announcement.

- Download and get started with .NET Core 3.1 on Windows, macOS, or Linux.

Long-term support

.NET Core 3.1 is an LTS release with support from Microsoft for three years after its release. It's highly recommended that you move your apps to the latest LTS release. See the .NET and .NET Core support policy page for a list of supported releases.

Release	End of life date
.NET Core 3.1	End of life on December 13, 2022.
.NET Core 3.0	End of life on March 3, 2020.
.NET Core 2.2	End of life on December 23, 2019.
.NET Core 2.1	End of life on August 21, 2021.

For more information, see the .NET and .NET Core support policy.

macOS appHost and notarization

macOS only

Starting with the notarized .NET Core SDK 3.1 for macOS, the appHost setting is disabled by default. For more information, see macOS Catalina Notarization and the impact on .NET Core downloads and projects.

When the appHost setting is enabled, .NET Core generates a native Mach-O executable when you build or publish. Your app runs in the context of the appHost when it is run from source code with the `dotnet run` command, or by starting the Mach-O executable directly.

Without the appHost, the only way a user can start a framework-dependent app is with the `dotnet <filename.dll>` command. An appHost is always created when you publish your app self-contained.

You can either configure the appHost at the project level, or toggle the appHost for a specific `dotnet` command with the `-p:UseAppHost` parameter:

- Project file

  XML

  ```xml
  <PropertyGroup>
    <UseAppHost>true</UseAppHost>
  </PropertyGroup>
  ```

- Command-line parameter

  .NET CLI

  ```
  dotnet run -p:UseAppHost=true
  ```

For more information about the `UseAppHost` setting, see MSBuild properties for Microsoft.NET.Sdk.

Windows Forms

Windows only

> **⚠ Warning**
>
> There are breaking changes in Windows Forms.

Legacy controls were included in Windows Forms that have been unavailable in the Visual Studio Designer Toolbox for some time. These were replaced with new controls back in .NET Framework 2.0. These have been removed from the Desktop SDK for .NET Core 3.1.

Removed control	Recommended replacement	Associated APIs removed
DataGrid	DataGridView	DataGridCell DataGridRow DataGridTableCollection DataGridColumnCollection DataGridTableStyle DataGridColumnStyle DataGridLineStyle DataGridParentRowsLabel DataGridParentRowsLabelStyle DataGridBoolColumn DataGridTextBox GridColumnStylesCollection GridTableStylesCollection HitTestType
ToolBar	ToolStrip	ToolBarAppearance
ToolBarButton	ToolStripButton	ToolBarButtonClickEventArgs ToolBarButtonClickEventHandler ToolBarButtonStyle ToolBarTextAlign
ContextMenu	ContextMenuStrip	
Menu	ToolStripDropDown ToolStripDropDownMenu	MenuItemCollection
MainMenu	MenuStrip	
MenuItem	ToolStripMenuItem	

We recommend you update your applications to .NET Core 3.1 and move to the replacement controls. Replacing the controls is a straightforward process, essentially "find and replace" on the type.

C++/CLI

Windows only

Support has been added for creating C++/CLI (also known as "managed C++") projects. Binaries produced from these projects are compatible with .NET Core 3.0 and later versions.

To add support for C++/CLI in Visual Studio 2019 version 16.4, install the Desktop development with C++ workload. This workload adds two templates to Visual Studio:

- CLR Class Library (.NET Core)
- CLR Empty Project (.NET Core)

Next steps

- Review the breaking changes between .NET Core 3.0 and 3.1.
- Review the breaking changes in .NET Core 3.1 for Windows Forms apps.

Breaking changes in .NET Core 3.1

Article • 07/28/2023

If you're migrating to version 3.1 of .NET Core or ASP.NET Core, the breaking changes listed in this article may affect your app.

ASP.NET Core

HTTP: Browser SameSite changes impact authentication

Some browsers, such as Chrome and Firefox, made breaking changes to their implementations of `SameSite` for cookies. The changes impact remote authentication scenarios, such as OpenID Connect and WS-Federation, which must opt out by sending `SameSite=None`. However, `SameSite=None` breaks on iOS 12 and some older versions of other browsers. The app needs to sniff these versions and omit `SameSite`.

For discussion on this issue, see dotnet/aspnetcore#14996.

Version introduced

3.1 Preview 1

Old behavior

`SameSite` is a 2016 draft standard extension to HTTP cookies. It's intended to mitigate Cross-Site Request Forgery (CSRF). This was originally designed as a feature the servers would opt into by adding the new parameters. ASP.NET Core 2.0 added initial support for `SameSite`.

New behavior

Google proposed a new draft standard that isn't backwards compatible. The standard changes the default mode to `Lax` and adds a new entry `None` to opt out. `Lax` suffices for most app cookies; however, it breaks cross-site scenarios like OpenID Connect and WS-Federation login. Most OAuth logins aren't affected because of differences in how the request flows. The new `None` parameter causes compatibility problems with clients that implemented the prior draft standard (for example, iOS 12). Chrome 80 will include the changes. See SameSite Updates for the Chrome product launch timeline.

ASP.NET Core 3.1 has been updated to implement the new `SameSite` behavior. The update redefines the behavior of `SameSiteMode.None` to emit `SameSite=None` and adds a new value `SameSiteMode.Unspecified` to omit the `SameSite` attribute. All cookie APIs now default to `Unspecified`, though some components that use cookies set values more specific to their scenarios such as the OpenID Connect correlation and nonce cookies.

For other recent changes in this area, see HTTP: Some cookie SameSite defaults changed to None. In ASP.NET Core 3.0, most defaults were changed from SameSiteMode.Lax to SameSiteMode.None (but still using the prior standard).

Reason for change

Browser and specification changes as outlined in the preceding text.

Recommended action

Apps that interact with remote sites, such as through third-party login, need to:

- Test those scenarios on multiple browsers.
- Apply the cookie policy browser sniffing mitigation discussed in Support older browsers.

For testing and browser sniffing instructions, see the following section.

Determine if you're affected

Test your web app using a client version that can opt into the new behavior. Chrome, Firefox, and Microsoft Edge Chromium all have new opt-in feature flags that can be used for testing. Verify that your app is compatible with older client versions after you've applied the patches, especially Safari. For more information, see Support older browsers.

Chrome

Chrome 78 and later yield misleading test results. Those versions have a temporary mitigation in place and allow cookies less than two minutes old. With the appropriate test flags enabled, Chrome 76 and 77 yield more accurate results. To test the new behavior, toggle `chrome://flags/#same-site-by-default-cookies` to enabled. Chrome 75 and earlier are reported to fail with the new `None` setting. For more information, see Support older browsers.

Google doesn't make older Chrome versions available. You can, however, download older versions of Chromium, which will suffice for testing. Follow the instructions at Download Chromium.

- Chromium 76 Win64
- Chromium 74 Win64

Safari

Safari 12 strictly implemented the prior draft and fails if it sees the new `None` value in cookies. This must be avoided via the browser sniffing code shown in Support older browsers. Ensure you test Safari 12 and 13 as well as WebKit-based, OS-style logins using Microsoft Authentication Library (MSAL), Active Directory Authentication Library (ADAL), or whichever library you're using. The problem is dependent on the underlying OS version. OSX Mojave 10.14 and iOS 12 are known to have compatibility problems with the new behavior. Upgrading to OSX Catalina 10.15 or iOS 13 fixes the problem. Safari doesn't currently have an opt-in flag for testing the new specification behavior.

Firefox

Firefox support for the new standard can be tested on version 68 and later by opting in on the `about:config` page with the feature flag `network.cookie.sameSite.laxByDefault`. No compatibility issues have been reported on older versions of Firefox.

Microsoft Edge

While Microsoft Edge supports the old `SameSite` standard, as of version 44 it didn't have any compatibility problems with the new standard.

Microsoft Edge Chromium

The feature flag is `edge://flags/#same-site-by-default-cookies`. No compatibility issues were observed when testing with Microsoft Edge Chromium 78.

Electron

Versions of Electron include older versions of Chromium. For example, the version of Electron used by Microsoft Teams is Chromium 66, which exhibits the older behavior. Perform your own compatibility testing with the version of Electron your product uses. For more information, see Support older browsers.

Support older browsers

The 2016 `SameSite` standard mandated that unknown values be treated as `SameSite=Strict` values. Consequently, any older browsers that support the original standard may break when they see a `SameSite` property with a value of `None`. Web apps must implement browser sniffing if they intend to support these old browsers. ASP.NET Core doesn't implement browser sniffing for you because `User-Agent` request header values are highly unstable and change on a weekly basis. Instead, an extension point in the cookie policy allows you to add `User-Agent`-specific logic.

In *Startup.cs*, add the following code:

C#

```
private void CheckSameSite(HttpContext httpContext, CookieOptions options)
{
    if (options.SameSite == SameSiteMode.None)
    {
        var userAgent = httpContext.Request.Headers["User-
Agent"].ToString();
        // TODO: Use your User Agent library of choice here.
        if (/* UserAgent doesn't support new behavior */)
        {
            options.SameSite = SameSiteMode.Unspecified;
        }
    }
}

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.MinimumSameSitePolicy = SameSiteMode.Unspecified;
        options.OnAppendCookie = cookieContext =>
            CheckSameSite(cookieContext.Context,
cookieContext.CookieOptions);
        options.OnDeleteCookie = cookieContext =>
            CheckSameSite(cookieContext.Context,
cookieContext.CookieOptions);
    });
}

public void Configure(IApplicationBuilder app)
{
    // Before UseAuthentication or anything else that writes cookies.
    app.UseCookiePolicy();

    app.UseAuthentication();
    // code omitted for brevity
}
```

Opt-out switches

The `Microsoft.AspNetCore.SuppressSameSiteNone` compatibility switch enables you to temporarily opt out of the new ASP.NET Core cookie behavior. Add the following JSON to a *runtimeconfig.template.json* file in your project:

JSON

```
{
  "configProperties": {
    "Microsoft.AspNetCore.SuppressSameSiteNone": "true"
  }
}
```

Other Versions

Related `SameSite` patches are forthcoming for:

- ASP.NET Core 2.1, 2.2, and 3.0
- `Microsoft.Owin` 4.1
- `System.Web` (for .NET Framework 4.7.2 and later)

Category

ASP.NET

Affected APIs

- Microsoft.AspNetCore.Builder.CookiePolicyOptions.MinimumSameSitePolicy
- Microsoft.AspNetCore.Http.CookieBuilder.SameSite
- Microsoft.AspNetCore.Http.CookieOptions.SameSite
- Microsoft.AspNetCore.Http.SameSiteMode
- Microsoft.Net.Http.Headers.SameSiteMode
- Microsoft.Net.Http.Headers.SetCookieHeaderValue.SameSite

Deployment

x86 host path on 64-bit Windows

MSBuild

Design-time builds only return top-level package references

Starting in .NET Core SDK 3.1.400, only top-level package references are returned by the `RunResolvePackageDependencies` target.

Version introduced

.NET Core SDK 3.1.400

Change description

In previous versions of the .NET Core SDK, the `RunResolvePackageDependencies` target created the following MSBuild items that contained information from the NuGet assets file:

- `PackageDefinitions`
- `PackageDependencies`
- `TargetDefinitions`
- `FileDefinitions`
- `FileDependencies`

This data is used by Visual Studio to populate the Dependencies node in Solution Explorer. However, it can be a large amount of data, and the data isn't needed unless the Dependencies node is expanded.

Starting in the .NET Core SDK version 3.1.400, most of these items aren't generated by default. Only items of type `Package` are returned. If Visual Studio needs the items to populate the Dependencies node, it reads the information directly from the assets file.

Reason for change

This changed was introduced to improve solution-load performance inside of Visual Studio. Previously, all package references would be loaded, which involved loading many references that most users would never view.

Recommended action

If you have MSBuild logic that depends on these items being created, set the `EmitLegacyAssetsFileItems` property to `true` in your project file. This setting enables the previous behavior where all the items are created.

Category

MSBuild

Affected APIs

N/A

SDK

Tool manifests in root folder

Windows Forms

Removed controls

Starting in .NET Core 3.1, some Windows Forms controls are no longer available.

Change description

Starting with .NET Core 3.1, various Windows Forms controls are no longer available. Replacement controls that have better design and support were introduced in .NET Framework 2.0. The deprecated controls were previously removed from designer toolboxes but were still available to be used.

The following types are no longer available:

- ContextMenu
- DataGrid
- DataGrid.HitTestType
- DataGridBoolColumn
- DataGridCell
- DataGridColumnStyle
- DataGridLineStyle
- DataGridParentRowsLabelStyle
- DataGridPreferredColumnWidthTypeConverter
- DataGridTableStyle
- DataGridTextBox
- DataGridTextBoxColumn
- GridColumnStylesCollection

- GridTablesFactory
- GridTableStylesCollection
- IDataGridEditingService
- IMenuEditorService
- MainMenu
- Menu
- Menu.MenuItemCollection
- MenuItem
- ToolBar
- ToolBarAppearance
- ToolBarButton
- ToolBar.ToolBarButtonCollection
- ToolBarButtonClickEventArgs
- ToolBarButtonStyle
- ToolBarTextAlign

Version introduced

3.1

Recommended action

Each removed control has a recommended replacement control. Refer to the following table:

Removed control (API)	Recommended replacement	Associated APIs that are removed
ContextMenu	ContextMenuStrip	
DataGrid	DataGridView	DataGridCell, DataGridRow, DataGridTableCollection, DataGridColumnCollection, DataGridTableStyle, DataGridColumnStyle, DataGridLineStyle, DataGridParentRowsLabel, DataGridParentRowsLabelStyle, DataGridBoolColumn, DataGridTextBox, GridColumnStylesCollection, GridTableStylesCollection, HitTestType
MainMenu	MenuStrip	
Menu	ToolStripDropDown, ToolStripDropDownMenu	MenuItemCollection

Removed control (API)	Recommended replacement	Associated APIs that are removed
MenuItem	ToolStripMenuItem	
ToolBar	ToolStrip	ToolBarAppearance
ToolBarButton	ToolStripButton	ToolBarButtonClickEventArgs, ToolBarButtonClickEventHandler, ToolBarButtonStyle, ToolBarTextAlign

Category

Windows Forms

Affected APIs

- System.Windows.Forms.ContextMenu
- System.Windows.Forms.GridColumnStylesCollection
- System.Windows.Forms.GridTablesFactory
- System.Windows.Forms.GridTableStylesCollection
- System.Windows.Forms.IDataGridEditingService
- System.Windows.Forms.MainMenu
- System.Windows.Forms.Menu
- System.Windows.Forms.Menu.MenuItemCollection
- System.Windows.Forms.MenuItem
- System.Windows.Forms.ToolBar
- System.Windows.Forms.ToolBar.ToolBarButtonCollection
- System.Windows.Forms.ToolBarAppearance
- System.Windows.Forms.ToolBarButton
- System.Windows.Forms.ToolBarButtonClickEventArgs
- System.Windows.Forms.ToolBarButtonStyle
- System.Windows.Forms.ToolBarTextAlign
- System.Windows.Forms.DataGrid
- System.Windows.Forms.DataGrid.HitTestType
- System.Windows.Forms.DataGridBoolColumn
- System.Windows.Forms.DataGridCell
- System.Windows.Forms.DataGridColumnStyle
- System.Windows.Forms.DataGridLineStyle
- System.Windows.Forms.DataGridParentRowsLabelStyle
- System.Windows.Forms.DataGridPreferredColumnWidthTypeConverter
- System.Windows.Forms.DataGridTableStyle

- System.Windows.Forms.DataGridTextBox
- System.Windows.Forms.DataGridTextBoxColumn
- System.Windows.Forms.Design.IMenuEditorService

CellFormatting event not raised if tooltip is shown

A DataGridView now shows a cell's text and error tooltips when hovered by a mouse and when selected via the keyboard. If a tooltip is shown, the DataGridView.CellFormatting event is not raised.

Change description

Prior to .NET Core 3.1, a DataGridView that had the ShowCellToolTips property set to `true` showed a tooltip for a cell's text and errors when the cell was hovered by a mouse. Tooltips were not shown when a cell was selected via the keyboard (for example, by using the Tab key, shortcut keys, or arrow navigation). If the user edited a cell, and then, while the DataGridView was still in edit mode, hovered over a cell that did not have the ToolTipText property set, a CellFormatting event was raised to format the cell's text for display in the cell.

To meet accessibility standards, starting in .NET Core 3.1, a DataGridView that has the ShowCellToolTips property set to `true` shows tooltips for a cell's text and errors not only when the cell is hovered, but also when it's selected via the keyboard. As a consequence of this change, the CellFormatting event is *not* raised when cells that don't have the ToolTipText property set are hovered while the DataGridView is in edit mode. The event is not raised because the content of the hovered cell is shown as a tooltip instead of being displayed in the cell.

Version introduced

3.1

Recommended action

Refactor any code that depends on the CellFormatting event while the DataGridView is in edit mode.

Category

Windows Forms

Affected APIs

None

See also

- What's new in .NET Core 3.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Core 3.0

Article • 02/13/2023

This article describes what is new in .NET Core 3.0. One of the biggest enhancements is support for Windows desktop applications (Windows only). By using the .NET Core 3.0 SDK component Windows Desktop, you can port your Windows Forms and Windows Presentation Foundation (WPF) applications. To be clear, the Windows Desktop component is only supported and included on Windows. For more information, see the Windows desktop section later in this article.

.NET Core 3.0 adds support for C# 8.0. It's highly recommended that you use Visual Studio 2019 version 16.3 or newer, Visual Studio for Mac 8.3 or newer, or Visual Studio Code with the latest **C# extension**.

Download and get started with .NET Core 3.0 right now on Windows, macOS, or Linux.

For more information about the release, see the .NET Core 3.0 announcement.

.NET Core 3.0 RC 1 was considered production ready by Microsoft and was fully supported. If you're using a preview release, you must move to the RTM version for continued support.

Language improvements C# 8.0

C# 8.0 is also part of this release, which includes the nullable reference types feature, async streams, and more patterns. For more information about C# 8.0 features, see What's new in C# 8.0.

Tutorials related to C# 8.0 language features:

- Tutorial: Express your design intent more clearly with nullable and non-nullable reference types
- Tutorial: Generate and consume async streams using C# 8.0 and .NET Core 3.0
- Tutorial: Use pattern matching to build type-driven and data-driven algorithms

Language enhancements were added to support the following API features detailed below:

- Ranges and indices
- Async streams

.NET Standard 2.1

.NET Core 3.0 implements **.NET Standard 2.1**. However, the default `dotnet new classlib` template generates a project that still targets **.NET Standard 2.0**. To target **.NET Standard 2.1**, edit your project file and change the `TargetFramework` property to `netstandard2.1`:

XML

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.1</TargetFramework>
  </PropertyGroup>

</Project>
```

If you're using Visual Studio, you need Visual Studio 2019, as Visual Studio 2017 doesn't support **.NET Standard 2.1** or **.NET Core 3.0**.

Compile/Deploy

Default executables

.NET Core now builds framework-dependent executables by default. This behavior is new for applications that use a globally installed version of .NET Core. Previously, only self-contained deployments would produce an executable.

During `dotnet build` or `dotnet publish`, an executable (known as the **appHost**) is created that matches the environment and platform of the SDK you're using. You can expect the same things with these executables as you would other native executables, such as:

- You can double-click on the executable.
- You can launch the application from a command prompt directly, such as `myapp.exe` on Windows, and `./myapp` on Linux and macOS.

macOS appHost and notarization

macOS only

Starting with the notarized .NET Core SDK 3.0 for macOS, the setting to produce a default executable (known as the appHost) is disabled by default. For more information, see macOS Catalina Notarization and the impact on .NET Core downloads and projects.

When the appHost setting is enabled, .NET Core generates a native Mach-O executable when you build or publish. Your app runs in the context of the appHost when it is run from source code with the `dotnet run` command, or by starting the Mach-O executable directly.

Without the appHost, the only way a user can start a framework-dependent app is with the `dotnet <filename.dll>` command. An appHost is always created when you publish your app self-contained.

You can either configure the appHost at the project level, or toggle the appHost for a specific `dotnet` command with the `-p:UseAppHost` parameter:

- Project file

  XML

  ```xml
  <PropertyGroup>
    <UseAppHost>true</UseAppHost>
  </PropertyGroup>
  ```

- Command-line parameter

  .NET CLI

  ```
  dotnet run -p:UseAppHost=true
  ```

For more information about the `UseAppHost` setting, see MSBuild properties for Microsoft.NET.Sdk.

Single-file executables

The `dotnet publish` command supports packaging your app into a platform-specific single-file executable. The executable is self-extracting and contains all dependencies (including native) that are required to run your app. When the app is first run, the application is extracted to a directory based on the app name and build identifier. Startup is faster when the application is run again. The application doesn't need to extract itself a second time unless a new version was used.

To publish a single-file executable, set the `PublishSingleFile` in your project or on the command line with the `dotnet publish` command:

XML

```xml
<PropertyGroup>
  <RuntimeIdentifier>win10-x64</RuntimeIdentifier>
  <PublishSingleFile>true</PublishSingleFile>
</PropertyGroup>
```

-or-

.NET CLI

```
dotnet publish -r win10-x64 -p:PublishSingleFile=true
```

For more information about single-file publishing, see the single-file bundler design document.

Assembly trimming

The .NET core 3.0 SDK comes with a tool that can reduce the size of apps by analyzing IL and trimming unused assemblies.

Self-contained apps include everything needed to run your code, without requiring .NET to be installed on the host computer. However, many times the app only requires a small subset of the framework to function, and other unused libraries could be removed.

.NET Core now includes a setting that will use the IL Trimmer tool to scan the IL of your app. This tool detects what code is required, and then trims unused libraries. This tool can significantly reduce the deployment size of some apps.

To enable this tool, add the `<PublishTrimmed>` setting in your project and publish a self-contained app:

XML

```xml
<PropertyGroup>
  <PublishTrimmed>true</PublishTrimmed>
</PropertyGroup>
```

.NET CLI

```
dotnet publish -r <rid> -c Release
```

As an example, the basic "hello world" new console project template that is included, when published, hits about 70 MB in size. By using `<PublishTrimmed>`, that size is reduced to about 30 MB.

It's important to consider that applications or frameworks (including ASP.NET Core and WPF) that use reflection or related dynamic features, will often break when trimmed. This breakage occurs because the trimmer doesn't know about this dynamic behavior and can't determine which framework types are required for reflection. The IL Trimmer tool can be configured to be aware of this scenario.

Above all else, be sure to test your app after trimming.

For more information about the IL Trimmer tool, see the documentation or visit the mono/linker repo.

Tiered compilation

Tiered compilation (TC) is on by default with .NET Core 3.0. This feature enables the runtime to more adaptively use the just-in-time (JIT) compiler to achieve better performance.

The main benefit of tiered compilation is to provide two ways of jitting methods: in a lower-quality-but-faster tier or a higher-quality-but-slower tier. The quality refers to how well the method is optimized. TC helps to improve the performance of an application as it goes through various stages of execution, from startup through steady state. When tiered compilation is disabled, every method is compiled in a single way that's biased to steady-state performance over startup performance.

When TC is enabled, the following behavior applies for method compilation when an app starts up:

- If the method has ahead-of-time-compiled code, or ReadyToRun, the pregenerated code is used.
- Otherwise, the method is jitted. Typically, these methods are generics over value types.
  - *Quick JIT* produces lower-quality (or less optimized) code more quickly. In .NET Core 3.0, Quick JIT is enabled by default for methods that don't contain loops and is preferred during startup.
  - The fully optimizing JIT produces higher-quality (or more optimized) code more slowly. For methods where Quick JIT would not be used (for example, if the

method is attributed with MethodImplOptions.AggressiveOptimization), the fully optimizing JIT is used.

For frequently called methods, the just-in-time compiler eventually creates fully optimized code in the background. The optimized code then replaces the pre-compiled code for that method.

Code generated by Quick JIT may run slower, allocate more memory, or use more stack space. If there are issues, you can disabled Quick JIT using this MSBuild property in the project file:

XML

```xml
<PropertyGroup>
  <TieredCompilationQuickJit>false</TieredCompilationQuickJit>
</PropertyGroup>
```

To disable TC completely, use this MSBuild property in your project file:

XML

```xml
<PropertyGroup>
  <TieredCompilation>false</TieredCompilation>
</PropertyGroup>
```

> **Tip**
>
> If you change these settings in the project file, you may need to perform a clean build for the new settings to be reflected (delete the `obj` and `bin` directories and rebuild).

For more information about configuring compilation at run time, see Runtime configuration options for compilation.

ReadyToRun images

You can improve the startup time of your .NET Core application by compiling your application assemblies as ReadyToRun (R2R) format. R2R is a form of ahead-of-time (AOT) compilation.

R2R binaries improve startup performance by reducing the amount of work the just-in-time (JIT) compiler needs to do as your application loads. The binaries contain similar native code compared to what the JIT would produce. However, R2R binaries are larger

because they contain both intermediate language (IL) code, which is still needed for some scenarios, and the native version of the same code. R2R is only available when you publish a self-contained app that targets specific runtime environments (RID) such as Linux x64 or Windows x64.

To compile your project as ReadyToRun, do the following:

1. Add the `<PublishReadyToRun>` setting to your project:

   XML

   ```xml
   <PropertyGroup>
     <PublishReadyToRun>true</PublishReadyToRun>
   </PropertyGroup>
   ```

2. Publish a self-contained app. For example, this command creates a self-contained app for the 64-bit version of Windows:

   .NET CLI

   ```
   dotnet publish -c Release -r win-x64 --self-contained
   ```

Cross platform/architecture restrictions

The ReadyToRun compiler doesn't currently support cross-targeting. You must compile on a given target. For example, if you want R2R images for Windows x64, you need to run the publish command on that environment.

Exceptions to cross-targeting:

- Windows x64 can be used to compile Windows Arm32, Arm64, and x86 images.
- Windows x86 can be used to compile Windows Arm32 images.
- Linux x64 can be used to compile Linux Arm32 and Arm64 images.

For more information, see Ready to Run.

Runtime/SDK

Major-version runtime roll forward

.NET Core 3.0 introduces an opt-in feature that allows your app to roll forward to the latest major version of .NET Core. Additionally, a new setting has been added to control

how roll forward is applied to your app. This can be configured in the following ways:

- Project file property: `RollForward`
- Runtime configuration file property: `rollForward`
- Environment variable: `DOTNET_ROLL_FORWARD`
- Command-line argument: `--roll-forward`

One of the following values must be specified. If the setting is omitted, **Minor** is the default.

- **LatestPatch**
  Roll forward to the highest patch version. This disables minor version roll forward.
- **Minor**
  Roll forward to the lowest higher minor version, if requested minor version is missing. If the requested minor version is present, then the **LatestPatch** policy is used.
- **Major**
  Roll forward to lowest higher major version, and lowest minor version, if requested major version is missing. If the requested major version is present, then the **Minor** policy is used.
- **LatestMinor**
  Roll forward to highest minor version, even if requested minor version is present. Intended for component hosting scenarios.
- **LatestMajor**
  Roll forward to highest major and highest minor version, even if requested major is present. Intended for component hosting scenarios.
- **Disable**
  Don't roll forward. Only bind to specified version. This policy isn't recommended for general use because it disables the ability to roll forward to the latest patches. This value is only recommended for testing.

Besides the **Disable** setting, all settings will use the highest available patch version.

By default, if the requested version (as specified in `.runtimeconfig.json` for the application) is a release version, only release versions are considered for roll forward. Any pre-release versions are ignored. If there is no matching release version, then pre-release versions are taken into account. This behavior can be changed by setting `DOTNET_ROLL_FORWARD_TO_PRERELEASE=1`, in which case all versions are always considered.

Build copies dependencies

The `dotnet build` command now copies NuGet dependencies for your application from the NuGet cache to the build output folder. Previously, dependencies were only copied as part of `dotnet publish`.

There are some operations, like trimming and razor page publishing, that will still require publishing.

Local tools

.NET Core 3.0 introduces local tools. Local tools are similar to global tools but are associated with a particular location on disk. Local tools aren't available globally and are distributed as NuGet packages.

Local tools rely on a manifest file name `dotnet-tools.json` in your current directory. This manifest file defines the tools to be available at that folder and below. You can distribute the manifest file with your code to ensure that anyone who works with your code can restore and use the same tools.

For both global and local tools, a compatible version of the runtime is required. Many tools currently on NuGet.org target .NET Core Runtime 2.1. To install these tools globally or locally, you would still need to install the NET Core 2.1 Runtime.

New global.json options

The *global.json* file has new options that provide more flexibility when you're trying to define which version of the .NET Core SDK is used. The new options are:

- `allowPrerelease`: Indicates whether the SDK resolver should consider prerelease versions when selecting the SDK version to use.
- `rollForward`: Indicates the roll-forward policy to use when selecting an SDK version, either as a fallback when a specific SDK version is missing or as a directive to use a higher version.

For more information about the changes including default values, supported values, and new matching rules, see global.json overview.

Smaller Garbage Collection heap sizes

The Garbage Collector's default heap size has been reduced resulting in .NET Core using less memory. This change better aligns with the generation 0 allocation budget with modern processor cache sizes.

Garbage Collection Large Page support

Large Pages (also known as Huge Pages on Linux) is a feature where the operating system is able to establish memory regions larger than the native page size (often 4K) to improve performance of the application requesting these large pages.

The Garbage Collector can now be configured with the **GCLargePages** setting as an opt-in feature to choose to allocate large pages on Windows.

Windows Desktop & COM

.NET Core SDK Windows Installer

The MSI installer for Windows has changed starting with .NET Core 3.0. The SDK installers will now upgrade SDK feature-band releases in place. Feature bands are defined in the *hundreds* groups in the *patch* section of the version number. For example, **3.0.*101*** and **3.0.*201*** are versions in two different feature bands while **3.0.*101*** and **3.0.*199*** are in the same feature band. And, when .NET Core SDK **3.0.*101*** is installed, .NET Core SDK **3.0.*100*** will be removed from the machine if it exists. When .NET Core SDK **3.0.*200*** is installed on the same machine, .NET Core SDK **3.0.*101*** won't be removed.

For more information about versioning, see Overview of how .NET Core is versioned.

Windows desktop

.NET Core 3.0 supports Windows desktop applications using Windows Presentation Foundation (WPF) and Windows Forms. These frameworks also support using modern controls and Fluent styling from the Windows UI XAML Library (WinUI) via XAML islands.

The Windows Desktop component is part of the Windows .NET Core 3.0 SDK.

You can create a new WPF or Windows Forms app with the following `dotnet` commands:

.NET CLI

```
dotnet new wpf
dotnet new winforms
```

Visual Studio 2019 adds **New Project** templates for .NET Core 3.0 Windows Forms and WPF.

For more information about how to port an existing .NET Framework application, see Port WPF projects and Port Windows Forms projects.

WinForms high DPI

.NET Core Windows Forms applications can set high DPI mode with Application.SetHighDpiMode(HighDpiMode). The `SetHighDpiMode` method sets the corresponding high DPI mode unless the setting has been set by other means like `App.Manifest` or P/Invoke before `Application.Run`.

The possible `highDpiMode` values, as expressed by the System.Windows.Forms.HighDpiMode enum are:

- `DpiUnaware`
- `SystemAware`
- `PerMonitor`
- `PerMonitorV2`
- `DpiUnawareGdiScaled`

For more information about high DPI modes, see High DPI Desktop Application Development on Windows.

Create COM components

On Windows, you can now create COM-callable managed components. This capability is critical to use .NET Core with COM add-in models and also to provide parity with .NET Framework.

Unlike .NET Framework where the *mscoree.dll* was used as the COM server, .NET Core will add a native launcher dll to the *bin* directory when you build your COM component.

For an example of how to create a COM component and consume it, see the COM Demo.

Windows Native Interop

Windows offers a rich native API in the form of flat C APIs, COM, and WinRT. While .NET Core supports **P/Invoke**, .NET Core 3.0 adds the ability to **CoCreate COM APIs** and **Activate WinRT APIs**. For a code example, see the Excel Demo.

MSIX Deployment

MSIX is a new Windows application package format. It can be used to deploy .NET Core 3.0 desktop applications to Windows 10.

The Windows Application Packaging Project, available in Visual Studio 2019, allows you to create MSIX packages with self-contained .NET Core applications.

The .NET Core project file must specify the supported runtimes in the `<RuntimeIdentifiers>` property:

```
XML

<RuntimeIdentifiers>win-x86;win-x64</RuntimeIdentifiers>
```

Linux improvements

SerialPort for Linux

.NET Core 3.0 provides basic support for System.IO.Ports.SerialPort on Linux.

Previously, .NET Core only supported using `SerialPort` on Windows.

For more information about the limited support for the serial port on Linux, see GitHub issue #33146.

Docker and cgroup memory Limits

Running .NET Core 3.0 on Linux with Docker works better with cgroup memory limits. Running a Docker container with memory limits, such as with `docker run -m`, changes how .NET Core behaves.

- Default Garbage Collector (GC) heap size: maximum of 20 mb or 75% of the memory limit on the container.
- Explicit size can be set as an absolute number or percentage of cgroup limit.
- Minimum reserved segment size per GC heap is 16 mb. This size reduces the number of heaps that are created on machines.

GPIO Support for Raspberry Pi

Two packages have been released to NuGet that you can use for GPIO programming:

- System.Device.Gpio
- Iot.Device.Bindings

The GPIO packages include APIs for *GPIO*, *SPI*, *I2C*, and *PWM* devices. The IoT bindings package includes device bindings. For more information, see the devices GitHub repo.

Arm64 Linux support

.NET Core 3.0 adds support for Arm64 for Linux. The primary use case for Arm64 is currently with IoT scenarios. For more information, see .NET Core Arm64 Status.

Docker images for .NET Core on Arm64 are available for Alpine, Debian, and Ubuntu.

> **Note**
>
> Support for the macOS Arm64 (or "Apple Silicon") and Windows Arm64 operating systems was later added in .NET 6.

Security

TLS 1.3 & OpenSSL 1.1.1 on Linux

.NET Core now takes advantage of TLS 1.3 support in OpenSSL 1.1.1, when it's available in a given environment. With TLS 1.3:

- Connection times are improved with reduced round trips required between the client and server.
- Improved security because of the removal of various obsolete and insecure cryptographic algorithms.

When available, .NET Core 3.0 uses **OpenSSL 1.1.1**, **OpenSSL 1.1.0**, or **OpenSSL 1.0.2** on a Linux system. When **OpenSSL 1.1.1** is available, both System.Net.Security.SslStream and System.Net.Http.HttpClient types will use **TLS 1.3** (assuming both the client and server support **TLS 1.3**).

> **Important**
>
> Windows and macOS do not yet support **TLS 1.3**.

The following C# 8.0 example demonstrates .NET Core 3.0 on Ubuntu 18.10 connecting to https://www.cloudflare.com:

```
C#
```

```
using System;
using System.Net.Security;
using System.Net.Sockets;
using System.Threading.Tasks;

namespace whats_new
{
    public static class TLS
    {
        public static async Task ConnectCloudFlare()
        {
            var targetHost = "www.cloudflare.com";

            using TcpClient tcpClient = new TcpClient();

            await tcpClient.ConnectAsync(targetHost, 443);

            using SslStream sslStream = new
SslStream(tcpClient.GetStream());

            await sslStream.AuthenticateAsClientAsync(targetHost);
            await Console.Out.WriteLineAsync($"Connected to {targetHost}
with {sslStream.SslProtocol}");
        }
    }
}
```

Cryptography ciphers

.NET Core 3.0 adds support for **AES-GCM** and **AES-CCM** ciphers, implemented with System.Security.Cryptography.AesGcm and System.Security.Cryptography.AesCcm respectively. These algorithms are both Authenticated Encryption with Association Data (AEAD) algorithms.

The following code demonstrates using `AesGcm` cipher to encrypt and decrypt random data.

C#

```
using System;
using System.Linq;
using System.Security.Cryptography;

namespace whats_new
{
    public static class Cipher
    {
        public static void Run()
        {
```

```
            // key should be: pre-known, derived, or transported via another
channel, such as RSA encryption
            byte[] key = new byte[16];
            RandomNumberGenerator.Fill(key);

            byte[] nonce = new byte[12];
            RandomNumberGenerator.Fill(nonce);

            // normally this would be your data
            byte[] dataToEncrypt = new byte[1234];
            byte[] associatedData = new byte[333];
            RandomNumberGenerator.Fill(dataToEncrypt);
            RandomNumberGenerator.Fill(associatedData);

            // these will be filled during the encryption
            byte[] tag = new byte[16];
            byte[] ciphertext = new byte[dataToEncrypt.Length];

            using (AesGcm aesGcm = new AesGcm(key))
            {
                aesGcm.Encrypt(nonce, dataToEncrypt, ciphertext, tag,
associatedData);
            }

            // tag, nonce, ciphertext, associatedData should be sent to the
other part

            byte[] decryptedData = new byte[ciphertext.Length];

            using (AesGcm aesGcm = new AesGcm(key))
            {
                aesGcm.Decrypt(nonce, ciphertext, tag, decryptedData,
associatedData);
            }

            // do something with the data
            // this should always print that data is the same
            Console.WriteLine($"AES-GCM: Decrypted data is
{(dataToEncrypt.SequenceEqual(decryptedData) ? "the same as" : "different
than")} original data.");
        }
    }
}
```

Cryptographic Key Import/Export

.NET Core 3.0 supports the import and export of asymmetric public and private keys from standard formats. You don't need to use an X.509 certificate.

All key types, such as *RSA*, *DSA*, *ECDsa*, and *ECDiffieHellman*, support the following formats:

- **Public Key**
  - X.509 SubjectPublicKeyInfo
- **Private key**
  - PKCS#8 PrivateKeyInfo
  - PKCS#8 EncryptedPrivateKeyInfo

RSA keys also support:

- **Public Key**
  - PKCS#1 RSAPublicKey
- **Private key**
  - PKCS#1 RSAPrivateKey

The export methods produce DER-encoded binary data, and the import methods expect the same. If a key is stored in the text-friendly PEM format, the caller will need to base64-decode the content before calling an import method.

C#

```
using System;
using System.Security.Cryptography;

namespace whats_new
{
    public static class RSATest
    {
        public static void Run(string keyFile)
        {
            using var rsa = RSA.Create();

            byte[] keyBytes = System.IO.File.ReadAllBytes(keyFile);
            rsa.ImportRSAPrivateKey(keyBytes, out int bytesRead);

            Console.WriteLine($"Read {bytesRead} bytes, {keyBytes.Length -
bytesRead} extra byte(s) in file.");
            RSAParameters rsaParameters = rsa.ExportParameters(true);
            Console.WriteLine(BitConverter.ToString(rsaParameters.D));
        }
    }
}
```

PKCS#8 files can be inspected with System.Security.Cryptography.Pkcs.Pkcs8PrivateKeyInfo and **PFX/PKCS#12** files can be inspected with System.Security.Cryptography.Pkcs.Pkcs12Info. **PFX/PKCS#12** files can be manipulated with System.Security.Cryptography.Pkcs.Pkcs12Builder.

.NET Core 3.0 API changes

Ranges and indices

The new System.Index type can be used for indexing. You can create one from an `int` that counts from the beginning, or with a prefix `^` operator (C#) that counts from the end:

C#

```
Index i1 = 3;  // number 3 from beginning
Index i2 = ^4; // number 4 from end
int[] a = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
Console.WriteLine($"{a[i1]}, {a[i2]}"); // "3, 6"
```

There's also the System.Range type, which consists of two `Index` values, one for the start and one for the end, and can be written with a `x..y` range expression (C#). You can then index with a `Range`, which produces a slice:

C#

```
var slice = a[i1..i2]; // { 3, 4, 5 }
```

For more information, see the ranges and indices tutorial.

Async streams

The IAsyncEnumerable<T> type is a new asynchronous version of IEnumerable<T>. The language lets you `await foreach` over `IAsyncEnumerable<T>` to consume their elements, and use `yield return` to them to produce elements.

The following example demonstrates both production and consumption of async streams. The `foreach` statement is async and itself uses `yield return` to produce an async stream for callers. This pattern (using `yield return`) is the recommended model for producing async streams.

C#

```
async IAsyncEnumerable<int> GetBigResultsAsync()
{
    await foreach (var result in GetResultsAsync())
    {
        if (result > 20) yield return result;
```

```
    }
}
```

In addition to being able to `await foreach`, you can also create async iterators, for example, an iterator that returns an `IAsyncEnumerable/IAsyncEnumerator` that you can both `await` and `yield` in. For objects that need to be disposed, you can use `IAsyncDisposable`, which various BCL types implement, such as `Stream` and `Timer`.

For more information, see the async streams tutorial.

IEEE Floating-point

Floating point APIs are being updated to comply with IEEE 754-2008 revision. The goal of these changes is to expose all **required** operations and ensure that they're behaviorally compliant with the IEEE spec. For more information about floating-point improvements, see the Floating-Point Parsing and Formatting improvements in .NET Core 3.0 blog post.

Parsing and formatting fixes include:

- Correctly parse and round inputs of any length.
- Correctly parse and format negative zero.
- Correctly parse `Infinity` and `NaN` by doing a case-insensitive check and allowing an optional preceding `+` where applicable.

New System.Math APIs include:

- BitIncrement(Double) and BitDecrement(Double)
  Corresponds to the `nextUp` and `nextDown` IEEE operations. They return the smallest floating-point number that compares greater or lesser than the input (respectively). For example, `Math.BitIncrement(0.0)` would return `double.Epsilon`.

- MaxMagnitude(Double, Double) and MinMagnitude(Double, Double)
  Corresponds to the `maxNumMag` and `minNumMag` IEEE operations, they return the value that is greater or lesser in magnitude of the two inputs (respectively). For example, `Math.MaxMagnitude(2.0, -3.0)` would return `-3.0`.

- ILogB(Double)
  Corresponds to the `logB` IEEE operation that returns an integral value, it returns the integral base-2 log of the input parameter. This method is effectively the same as `floor(log2(x))`, but done with minimal rounding error.

- ScaleB(Double, Int32)
  Corresponds to the `scaleB` IEEE operation that takes an integral value, it returns effectively `x * pow(2, n)`, but is done with minimal rounding error.

- Log2(Double)
  Corresponds to the `log2` IEEE operation, it returns the base-2 logarithm. It minimizes rounding error.

- FusedMultiplyAdd(Double, Double, Double)
  Corresponds to the `fma` IEEE operation, it performs a fused multiply add. That is, it does `(x * y) + z` as a single operation, thereby minimizing the rounding error. An example is `FusedMultiplyAdd(1e308, 2.0, -1e308)`, which returns `1e308`. The regular `(1e308 * 2.0) - 1e308` returns `double.PositiveInfinity`.

- CopySign(Double, Double)
  Corresponds to the `copySign` IEEE operation, it returns the value of `x`, but with the sign of `y`.

.NET Platform-Dependent Intrinsics

APIs have been added that allow access to certain perf-oriented CPU instructions, such as the **SIMD** or **Bit Manipulation instruction** sets. These instructions can help achieve significant performance improvements in certain scenarios, such as processing data efficiently in parallel.

Where appropriate, the .NET libraries have begun using these instructions to improve performance.

For more information, see .NET Platform-Dependent Intrinsics.

Improved .NET Core Version APIs

Starting with .NET Core 3.0, the version APIs provided with .NET Core now return the information you expect. For example:

```
C#

System.Console.WriteLine($"Environment.Version:
{System.Environment.Version}");

// Old result
//   Environment.Version: 4.0.30319.42000
//
```

```
// New result
//   Environment.Version: 3.0.0
```

C#

```
System.Console.WriteLine($"RuntimeInformation.FrameworkDescription:
{System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription}");

// Old result
//   RuntimeInformation.FrameworkDescription: .NET Core 4.6.27415.71
//
// New result (notice the value includes any preview release information)
//   RuntimeInformation.FrameworkDescription: .NET Core 3.0.0-preview4-
27615-11
```

> ⚠ **Warning**
>
> Breaking change. This is technically a breaking change because the versioning scheme has changed.

Fast built-in JSON support

.NET users have largely relied on Newtonsoft.Json and other popular JSON libraries, which continue to be good choices. `Newtonsoft.Json` uses .NET strings as its base datatype, which is UTF-16 under the hood.

The new built-in JSON support is high-performance, low allocation, and works with UTF-8 encoded JSON text. For more information about the System.Text.Json namespace and types, see the following articles:

- JSON serialization in .NET - overview
- How to serialize and deserialize JSON in .NET.
- How to migrate from Newtonsoft.Json to System.Text.Json

HTTP/2 support

The System.Net.Http.HttpClient type supports the HTTP/2 protocol. If HTTP/2 is enabled, the HTTP protocol version is negotiated via TLS/ALPN, and HTTP/2 is used if the server elects to use it.

The default protocol remains HTTP/1.1, but HTTP/2 can be enabled in two different ways. First, you can set the HTTP request message to use HTTP/2:

C#

```
var client = new HttpClient() { BaseAddress = new
Uri("https://localhost:5001") };

// HTTP/1.1 request
using (var response = await client.GetAsync("/"))
    Console.WriteLine(response.Content);

// HTTP/2 request
using (var request = new HttpRequestMessage(HttpMethod.Get, "/") { Version =
new Version(2, 0) })
using (var response = await client.SendAsync(request))
    Console.WriteLine(response.Content);
```

Second, you can change HttpClient to use HTTP/2 by default:

C#

```
var client = new HttpClient()
{
    BaseAddress = new Uri("https://localhost:5001"),
    DefaultRequestVersion = new Version(2, 0)
};

// HTTP/2 is default
using (var response = await client.GetAsync("/"))
    Console.WriteLine(response.Content);
```

Many times when you're developing an application, you want to use an unencrypted connection. If you know the target endpoint will be using HTTP/2, you can turn on unencrypted connections for HTTP/2. You can turn it on by setting the `DOTNET_SYSTEM_NET_HTTP_SOCKETSHTTPHANDLER_HTTP2UNENCRYPTEDSUPPORT` environment variable to `1` or by enabling it in the app context:

C#

```
AppContext.SetSwitch("System.Net.Http.SocketsHttpHandler.Http2UnencryptedSup
port", true);
```

Next steps

- Review the breaking changes between .NET Core 2.2 and 3.0.
- Review the breaking changes in .NET Core 3.0 for Windows Forms apps.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

Breaking changes in .NET Core 3.0

Article • 03/18/2023

If you're migrating to version 3.0 of .NET Core, ASP.NET Core, or EF Core, the breaking changes listed in this article may affect your app.

ASP.NET Core

- Obsolete Antiforgery, CORS, Diagnostics, MVC, and Routing APIs removed
- "Pubternal" APIs removed
- Authentication: Google+ deprecation
- Authentication: HttpContext.Authentication property removed
- Authentication: Newtonsoft.Json types replaced
- Authentication: OAuthHandler ExchangeCodeAsync signature changed
- Authorization: AddAuthorization overload moved to different assembly
- Authorization: IAllowAnonymous removed from AuthorizationFilterContext.Filters
- Authorization: IAuthorizationPolicyProvider implementations require new method
- Caching: CompactOnMemoryPressure property removed
- Caching: Microsoft.Extensions.Caching.SqlServer uses new SqlClient package
- Caching: ResponseCaching "pubternal" types changed to internal
- Data Protection: DataProtection.Blobs uses new Azure Storage APIs
- Hosting: AspNetCoreModule V1 removed from Windows Hosting Bundle
- Hosting: Generic host restricts Startup constructor injection
- Hosting: HTTPS redirection enabled for IIS out-of-process apps
- Hosting: IHostingEnvironment and IApplicationLifetime types replaced
- Hosting: ObjectPoolProvider removed from WebHostBuilder dependencies
- HTTP: DefaultHttpContext extensibility removed
- HTTP: HeaderNames fields changed to static readonly
- HTTP: Response body infrastructure changes
- HTTP: Some cookie SameSite default values changed
- HTTP: Synchronous IO disabled by default
- Identity: AddDefaultUI method overload removed
- Identity: UI Bootstrap version change
- Identity: SignInAsync throws exception for unauthenticated identity
- Identity: SignInManager constructor accepts new parameter
- Identity: UI uses static web assets feature
- Kestrel: Connection adapters removed
- Kestrel: Empty HTTPS assembly removed
- Kestrel: Request trailer headers moved to new collection

- Kestrel: Transport abstraction layer changes
- Localization: APIs marked obsolete
- Logging: DebugLogger class made internal
- MVC: Controller action Async suffix removed
- MVC: JsonResult moved to Microsoft.AspNetCore.Mvc.Core
- MVC: Precompilation tool deprecated
- MVC: Types changed to internal
- MVC: Web API compatibility shim removed
- Razor: RazorTemplateEngine API removed
- Razor: Runtime compilation moved to a package
- Session state: Obsolete APIs removed
- Shared framework: Assembly removal from Microsoft.AspNetCore.App
- Shared framework: Microsoft.AspNetCore.All removed
- SignalR: HandshakeProtocol.SuccessHandshakeData replaced
- SignalR: HubConnection methods removed
- SignalR: HubConnectionContext constructors changed
- SignalR: JavaScript client package name change
- SignalR: Obsolete APIs
- SPAs: SpaServices and NodeServices marked obsolete
- SPAs: SpaServices and NodeServices console logger fallback default change
- Target framework: .NET Framework not supported

Obsolete Antiforgery, CORS, Diagnostics, MVC, and Routing APIs removed

Obsolete members and compatibility switches in ASP.NET Core 2.2 were removed.

Version introduced

3.0

Reason for change

Improvement of API surface over time.

Recommended action

While targeting .NET Core 2.2, follow the guidance in the obsolete build messages to adopt new APIs instead.

Category

ASP.NET Core

Affected APIs

The following types and members were marked as obsolete for ASP.NET Core 2.1 and 2.2:

Types

- `Microsoft.AspNetCore.Diagnostics.Views.WelcomePage`
- `Microsoft.AspNetCore.DiagnosticsViewPage.Views.AttributeValue`
- `Microsoft.AspNetCore.DiagnosticsViewPage.Views.BaseView`
- `Microsoft.AspNetCore.DiagnosticsViewPage.Views.HelperResult`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.ProblemDetails21Wrapper`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.ValidationProblemDetails21Wrapper`
- `Microsoft.AspNetCore.Mvc.Razor.Compilation.ViewsFeatureProvider`
- `Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.PageArgumentBinder`
- `Microsoft.AspNetCore.Routing.IRouteValuesAddressMetadata`
- `Microsoft.AspNetCore.Routing.RouteValuesAddressMetadata`

Constructors

- `Microsoft.AspNetCore.Cors.Infrastructure.CorsService(IOptions{CorsOptions})`
- `Microsoft.AspNetCore.Routing.Tree.TreeRouteBuilder(ILoggerFactory,UrlEncoder,ObjectPool{UriBuildingContext},IInlineConstraintResolver)`
- `Microsoft.AspNetCore.Mvc.Formatters.OutputFormatterCanWriteContext`
- `Microsoft.AspNetCore.Mvc.ApiExplorer.DefaultApiDescriptionProvider(IOptions{MvcOptions},IInlineConstraintResolver,IModelMetadataProvider)`
- `Microsoft.AspNetCore.Mvc.ApiExplorer.DefaultApiDescriptionProvider(IOptions{MvcOptions},IInlineConstraintResolver,IModelMetadataProvider,IActionResultTypeMapper)`
- `Microsoft.AspNetCore.Mvc.Formatters.FormatFilter(IOptions{MvcOptions})`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ArrayModelBinder`1(IModelBinder)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ByteArrayModelBinder`
- Microsoft.AspNetCore.Mvc.ModelBinding.Binders.CollectionModelBinder`1(IModelBinder)

- Microsoft.AspNetCore.Mvc.ModelBinding.Binders.ComplexTypeModelBinder(IDictionary`2)
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.DictionaryModelBinder`2(IModelBinder,IModelBinder)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.DoubleModelBinder(System.Globalization.NumberStyles)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FloatModelBinder(System.Globalization.NumberStyles)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormCollectionModelBinder`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.FormFileModelBinder`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.HeaderModelBinder`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.KeyValuePairModelBinder`2(IModelBinder,IModelBinder)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Binders.SimpleTypeModelBinder(System.Type)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ModelAttributes(IEnumerable{System.Object})`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ModelAttributes(IEnumerable{System.Object},IEnumerable{System.Object})`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ModelBinderFactory(IModelMetadataProvider,IOptions{MvcOptions})`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder(IModelMetadataProvider,IModelBinderFactory,IObjectModelValidator)`
- Microsoft.AspNetCore.Mvc.Routing.KnownRouteValueConstraint()
- `Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter`
- `Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter(System.Boolean)`
- `Microsoft.AspNetCore.Mvc.Formatters.XmlDataContractSerializerInputFormatter(MvcOptions)`
- `Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter`
- `Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter(System.Boolean)`
- `Microsoft.AspNetCore.Mvc.Formatters.XmlSerializerInputFormatter(MvcOptions)`
- Microsoft.AspNetCore.Mvc.TagHelpers.ImageTagHelper(IHostingEnvironment,IMemoryCache,HtmlEncoder,IUrlHelperFactory)
- `Microsoft.AspNetCore.Mvc.TagHelpers.LinkTagHelper(IHostingEnvironment,IMemoryCache,HtmlEncoder,JavaScriptEncoder,IUrlHelperFactory)`

- `Microsoft.AspNetCore.Mvc.TagHelpers.ScriptTagHelper(IHostingEnvironment,IMemoryCache,HtmlEncoder,JavaScriptEncoder,IUrlHelperFactory)`
- `Microsoft.AspNetCore.Mvc.RazorPages.Infrastructure.RazorPageAdapter(RazorPageBase)`

Properties

- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookieDomain`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookieName`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.CookiePath`
- `Microsoft.AspNetCore.Antiforgery.AntiforgeryOptions.RequireSsl`
- `Microsoft.AspNetCore.Mvc.ApiBehaviorOptions.AllowInferringBindingSourceForCollectionTypesAsFromQuery`
- `Microsoft.AspNetCore.Mvc.ApiBehaviorOptions.SuppressUseValidationProblemDetailsForInvalidModelStateResponses`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.CookieName`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.Domain`
- `Microsoft.AspNetCore.Mvc.CookieTempDataProviderOptions.Path`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.MvcDataAnnotationsLocalizationOptions.AllowDataAnnotationsLocalizationForEnumDisplayAttributes`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.MvcXmlOptions.AllowRfc7807CompliantProblemDetailsFormat`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowBindingHeaderValuesToNonStringModelTypes`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowCombiningAuthorizeFilters`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowShortCircuitingValidationWhenNoValidatorsArePresent`
- `Microsoft.AspNetCore.Mvc.MvcOptions.AllowValidatingTopLevelNodes`
- `Microsoft.AspNetCore.Mvc.MvcOptions.InputFormatterExceptionPolicy`
- `Microsoft.AspNetCore.Mvc.MvcOptions.SuppressBindingUndefinedValueToEnumType`
- `Microsoft.AspNetCore.Mvc.MvcViewOptions.AllowRenderingMaxLengthAttribute`
- `Microsoft.AspNetCore.Mvc.MvcViewOptions.SuppressTempDataAttributePrefix`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowAreas`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowDefaultHandlingForOptionsRequests`
- `Microsoft.AspNetCore.Mvc.RazorPages.RazorPagesOptions.AllowMappingHeadRequestsToGetHandler`

Methods

- `Microsoft.AspNetCore.Mvc.LocalRedirectResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToActionResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToPageResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.RedirectToRouteResult.ExecuteResult(ActionContext)`
- `Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder.BindModelAsync(ActionContext,IValueProvider,ParameterDescriptor)`
- Microsoft.AspNetCore.Mvc.ModelBinding.ParameterBinder.BindModelAsync(ActionContext,IValueProvider,ParameterDescriptor,Object)

"Pubternal" APIs removed

To better maintain the public API surface of ASP.NET Core, most of the types in `*.Internal` namespaces (referred to as "pubternal" APIs) have become truly internal. Members in these namespaces were never meant to be supported as public-facing APIs. The APIs could break in minor releases and often did. Code that depends on these APIs breaks when updating to ASP.NET Core 3.0.

For more information, see dotnet/aspnetcore#4932 and dotnet/aspnetcore#11312.

Version introduced

3.0

Old behavior

The affected APIs are marked with the `public` access modifier and exist in `*.Internal` namespaces.

New behavior

The affected APIs are marked with the internal access modifier and can no longer be used by code outside that assembly.

Reason for change

The guidance for these "pubternal" APIs was that they:

- Could change without notice.
- Weren't subject to .NET policies to prevent breaking changes.

Leaving the APIs `public` (even in the `*.Internal` namespaces) was confusing to customers.

Recommended action

Stop using these "pubternal" APIs. If you have questions about alternate APIs, open an issue in the dotnet/aspnetcore repository.

For example, consider the following HTTP request buffering code in an ASP.NET Core 2.2 project. The `EnableRewind` extension method exists in the `Microsoft.AspNetCore.Http.Internal` namespace.

```
C#

HttpContext.Request.EnableRewind();
```

In an ASP.NET Core 3.0 project, replace the `EnableRewind` call with a call to the `EnableBuffering` extension method. The request buffering feature works as it did in the past. `EnableBuffering` calls the now `internal` API.

```
C#

HttpContext.Request.EnableBuffering();
```

Category

ASP.NET Core

Affected APIs

All APIs in the `Microsoft.AspNetCore.*` and `Microsoft.Extensions.*` namespaces that have an `Internal` segment in the namespace name. For example:

- `Microsoft.AspNetCore.Authentication.Internal`
- `Microsoft.AspNetCore.Builder.Internal`
- `Microsoft.AspNetCore.DataProtection.Cng.Internal`
- `Microsoft.AspNetCore.DataProtection.Internal`
- `Microsoft.AspNetCore.Hosting.Internal`
- `Microsoft.AspNetCore.Http.Internal`
- `Microsoft.AspNetCore.Mvc.Core.Infrastructure`

- `Microsoft.AspNetCore.Mvc.Core.Internal`
- `Microsoft.AspNetCore.Mvc.Cors.Internal`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Json.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.Internal`
- `Microsoft.AspNetCore.Mvc.Internal`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Internal`
- `Microsoft.AspNetCore.Mvc.Razor.Internal`
- `Microsoft.AspNetCore.Mvc.RazorPages.Internal`
- `Microsoft.AspNetCore.Mvc.TagHelpers.Internal`
- `Microsoft.AspNetCore.Mvc.ViewFeatures.Internal`
- `Microsoft.AspNetCore.Rewrite.Internal`
- `Microsoft.AspNetCore.Routing.Internal`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Adapter.Internal`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http`
- `Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Infrastructure`
- `Microsoft.AspNetCore.Server.Kestrel.Https.Internal`

Authentication: Google+ deprecated and replaced

Google is starting to shut down Google+ Sign-in for apps as early as January 28, 2019.

Change description

ASP.NET 4.x and ASP.NET Core have been using the Google+ Sign-in APIs to authenticate Google account users in web apps. The affected NuGet packages are Microsoft.AspNetCore.Authentication.Google for ASP.NET Core and Microsoft.Owin.Security.Google for `Microsoft.Owin` with ASP.NET Web Forms and MVC.

Google's replacement APIs use a different data source and format. The mitigations and solutions provided below account for the structural changes. Apps should verify the data itself still satisfies their requirements. For example, names, email addresses, profile links, and profile photos may provide subtly different values than before.

Version introduced

All versions. This change is external to ASP.NET Core.

Recommended action

Owin with ASP.NET Web Forms and MVC

For `Microsoft.Owin` 3.1.0 and later, a temporary mitigation is outlined here. Apps should complete testing with the mitigation to check for changes in the data format. There are plans to release `Microsoft.Owin` 4.0.1 with a fix. Apps using any prior version should update to version 4.0.1.

ASP.NET Core 1.x

The mitigation in Owin with ASP.NET Web Forms and MVC can be adapted to ASP.NET Core 1.x. NuGet package patches aren't planned because 1.x has reached end of life status.

ASP.NET Core 2.x

For `Microsoft.AspNetCore.Authentication.Google` version 2.x, replace your existing call to `AddGoogle` in `Startup.ConfigureServices` with the following code:

C#

```
.AddGoogle(o =>
{
    o.ClientId = Configuration["Authentication:Google:ClientId"];
    o.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
    o.UserInformationEndpoint =
"https://www.googleapis.com/oauth2/v2/userinfo";
    o.ClaimActions.Clear();
    o.ClaimActions.MapJsonKey(ClaimTypes.NameIdentifier, "id");
    o.ClaimActions.MapJsonKey(ClaimTypes.Name, "name");
    o.ClaimActions.MapJsonKey(ClaimTypes.GivenName, "given_name");
    o.ClaimActions.MapJsonKey(ClaimTypes.Surname, "family_name");
    o.ClaimActions.MapJsonKey("urn:google:profile", "link");
    o.ClaimActions.MapJsonKey(ClaimTypes.Email, "email");
});
```

The February 2.1 and 2.2 patches incorporated the preceding reconfiguration as the new default. No patch is planned for ASP.NET Core 2.0 since it has reached end of life.

ASP.NET Core 3.0

The mitigation given for ASP.NET Core 2.x can also be used for ASP.NET Core 3.0. In future 3.0 previews, the `Microsoft.AspNetCore.Authentication.Google` package may be removed. Users would be directed to `Microsoft.AspNetCore.Authentication.OpenIdConnect` instead. The following code shows how to replace `AddGoogle` with `AddOpenIdConnect` in `Startup.ConfigureServices`. This replacement can be used with ASP.NET Core 2.0 and later and can be adapted for ASP.NET Core 1.x as needed.

C#

```
.AddOpenIdConnect("Google", o =>
{
    o.ClientId = Configuration["Authentication:Google:ClientId"];
    o.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
    o.Authority = "https://accounts.google.com";
    o.ResponseType = OpenIdConnectResponseType.Code;
    o.CallbackPath = "/signin-google"; // Or register the default "/signin-
oidc"
    o.Scope.Add("email");
});
JwtSecurityTokenHandler.DefaultInboundClaimTypeMap.Clear();
```

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Authentication.Google

Authentication: HttpContext.Authentication property removed

The deprecated `Authentication` property on `HttpContext` has been removed.

Change description

As part of dotnet/aspnetcore#6504, the deprecated `Authentication` property on `HttpContext` has been removed. The `Authentication` property has been deprecated since 2.0. A migration guide was published to migrate code using this deprecated property to the new replacement APIs. The remaining unused classes / APIs related to

the old ASP.NET Core 1.x authentication stack were removed in commit dotnet/aspnetcore@d7a7c65 .

For discussion, see dotnet/aspnetcore#6533 .

Version introduced

3.0

Reason for change

ASP.NET Core 1.0 APIs have been replaced by extension methods in Microsoft.AspNetCore.Authentication.AuthenticationHttpContextExtensions.

Recommended action

See the migration guide.

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Http.Authentication.AuthenticateInfo
- Microsoft.AspNetCore.Http.Authentication.AuthenticationManager
- Microsoft.AspNetCore.Http.Authentication.AuthenticationProperties
- Microsoft.AspNetCore.Http.Features.Authentication.AuthenticateContext
- Microsoft.AspNetCore.Http.Features.Authentication.ChallengeBehavior
- Microsoft.AspNetCore.Http.Features.Authentication.ChallengeContext
- Microsoft.AspNetCore.Http.Features.Authentication.DescribeSchemesContext
- Microsoft.AspNetCore.Http.Features.Authentication.IAuthenticationHandler
- Microsoft.AspNetCore.Http.Features.Authentication.IHttpAuthenticationFeature.Handler
- Microsoft.AspNetCore.Http.Features.Authentication.SignInContext
- Microsoft.AspNetCore.Http.Features.Authentication.SignOutContext
- Microsoft.AspNetCore.Http.HttpContext.Authentication

Authentication: Newtonsoft.Json types replaced

In ASP.NET Core 3.0, `Newtonsoft.Json` types used in Authentication APIs have been replaced with `System.Text.Json` types. Except for the following cases, basic usage of the Authentication packages remains unaffected:

- Classes derived from the OAuth providers, such as those from aspnet-contrib.
- Advanced claim manipulation implementations.

For more information, see dotnet/aspnetcore#7105. For discussion, see dotnet/aspnetcore#7289.

Version introduced

3.0

Recommended action

For derived OAuth implementations, the most common change is to replace `JObject.Parse` with `JsonDocument.Parse` in the `CreateTicketAsync` override as shown here. `JsonDocument` implements `IDisposable`.

The following list outlines known changes:

- ClaimAction.Run(JObject, ClaimsIdentity, String) becomes `ClaimAction.Run(JsonElement userData, ClaimsIdentity identity, string issuer)`. All derived implementations of `ClaimAction` are similarly affected.
- ClaimActionCollectionMapExtensions.MapCustomJson(ClaimActionCollection, String, Func<JObject,String>) becomes `MapCustomJson(this ClaimActionCollection collection, string claimType, Func<JsonElement, string> resolver)`
- ClaimActionCollectionMapExtensions.MapCustomJson(ClaimActionCollection, String, String, Func<JObject,String>) becomes `MapCustomJson(this ClaimActionCollection collection, string claimType, string valueType, Func<JsonElement, string> resolver)`
- OAuthCreatingTicketContext has had one old constructor removed and the other replaced `JObject` with `JsonElement`. The `User` property and `RunClaimActions` method have been updated to match.
- Success(JObject) now accepts a parameter of type `JsonDocument` instead of `JObject`. The `Response` property has been updated to match. `OAuthTokenResponse` is now disposable and will be disposed by `OAuthHandler`. Derived OAuth implementations overriding `ExchangeCodeAsync` don't need to dispose the `JsonDocument` or `OAuthTokenResponse`.

- UserInformationReceivedContext.User changed from `JObject` to `JsonDocument`.
- TwitterCreatingTicketContext.User changed from `JObject` to `JsonElement`.
- The last parameter of TwitterHandler.CreateTicketAsync(ClaimsIdentity,AuthenticationProperties,AccessToken,JObject) changed from `JObject` to `JsonElement`. The replacement method is TwitterHandler.CreateTicketAsync(ClaimsIdentity, AuthenticationProperties, AccessToken, JsonElement).

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Authentication.Facebook
- Microsoft.AspNetCore.Authentication.Google
- Microsoft.AspNetCore.Authentication.MicrosoftAccount
- Microsoft.AspNetCore.Authentication.OAuth
- Microsoft.AspNetCore.Authentication.OpenIdConnect
- Microsoft.AspNetCore.Authentication.Twitter

Authentication: OAuthHandler ExchangeCodeAsync signature changed

In ASP.NET Core 3.0, the signature of `OAuthHandler.ExchangeCodeAsync` was changed from:

C#

```
protected virtual
System.Threading.Tasks.Task<Microsoft.AspNetCore.Authentication.OAuth.OAuthTokenResponse> ExchangeCodeAsync(string code, string redirectUri) { throw null; }
```

To:

C#

```
protected virtual
System.Threading.Tasks.Task<Microsoft.AspNetCore.Authentication.OAuth.OAuthTokenResponse>
```

```
ExchangeCodeAsync(Microsoft.AspNetCore.Authentication.OAuth.OAuthCodeExchang
eContext context) { throw null; }
```

Version introduced

3.0

Old behavior

The `code` and `redirectUri` strings were passed as separate arguments.

New behavior

`Code` and `RedirectUri` are properties on `OAuthCodeExchangeContext` that can be set via the `OAuthCodeExchangeContext` constructor. The new `OAuthCodeExchangeContext` type is the only argument passed to `OAuthHandler.ExchangeCodeAsync`.

Reason for change

This change allows additional parameters to be provided in a non-breaking manner. There's no need to create new `ExchangeCodeAsync` overloads.

Recommended action

Construct an `OAuthCodeExchangeContext` with the appropriate `code` and `redirectUri` values. An AuthenticationProperties instance must be provided. This single `OAuthCodeExchangeContext` instance can be passed to `OAuthHandler.ExchangeCodeAsync` instead of multiple arguments.

Category

ASP.NET Core

Affected APIs

OAuthHandler<TOptions>.ExchangeCodeAsync(String, String)

Authorization: AddAuthorization overload moved to different assembly

The core `AddAuthorization` methods that used to reside in `Microsoft.AspNetCore.Authorization` were renamed to `AddAuthorizationCore`. The old `AddAuthorization` methods still exist, but are in the `Microsoft.AspNetCore.Authorization.Policy` assembly instead. Apps using both methods should see no impact. Note that `Microsoft.AspNetCore.Authorization.Policy` now ships in the shared framework rather than a standalone package as discussed in Shared framework: Assemblies removed from Microsoft.AspNetCore.App.

Version introduced

3.0

Old behavior

`AddAuthorization` methods existed in `Microsoft.AspNetCore.Authorization`.

New behavior

`AddAuthorization` methods exist in `Microsoft.AspNetCore.Authorization.Policy`. `AddAuthorizationCore` is the new name for the old methods.

Reason for change

`AddAuthorization` is a better method name for adding all common services needed for authorization.

Recommended action

Either add a reference to `Microsoft.AspNetCore.Authorization.Policy` or use `AddAuthorizationCore` instead.

Category

ASP.NET Core

Affected APIs

Microsoft.Extensions.DependencyInjection.AuthorizationServiceCollectionExtensions.AddAuthorization(IServiceCollection, Action<AuthorizationOptions>)

Authorization: IAllowAnonymous removed from AuthorizationFilterContext.Filters

As of ASP.NET Core 3.0, MVC doesn't add AllowAnonymousFilters for [AllowAnonymous] attributes that were discovered on controllers and action methods. This change is addressed locally for derivatives of AuthorizeAttribute, but it's a breaking change for IAsyncAuthorizationFilter and IAuthorizationFilter implementations. Such implementations wrapped in a [TypeFilter] attribute are a popular and supported way to achieve strongly-typed, attribute-based authorization when both configuration and dependency injection are required.

Version introduced

3.0

Old behavior

IAllowAnonymous appeared in the AuthorizationFilterContext.Filters collection. Testing for the interface's presence was a valid approach to override or disable the filter on individual controller methods.

New behavior

`IAllowAnonymous` no longer appears in the `AuthorizationFilterContext.Filters` collection. `IAsyncAuthorizationFilter` implementations that are dependent on the old behavior typically cause intermittent HTTP 401 Unauthorized or HTTP 403 Forbidden responses.

Reason for change

A new endpoint routing strategy was introduced in ASP.NET Core 3.0.

Recommended action

Search the endpoint metadata for `IAllowAnonymous`. For example:

C#

```
var endpoint = context.HttpContext.GetEndpoint();
if (endpoint?.Metadata?.GetMetadata<IAllowAnonymous>() != null)
```

```
{
}
```

An example of this technique is seen in this HasAllowAnonymous method.

Category

ASP.NET Core

Affected APIs

None

Authorization: IAuthorizationPolicyProvider implementations require new method

In ASP.NET Core 3.0, a new `GetFallbackPolicyAsync` method was added to `IAuthorizationPolicyProvider`. This fallback policy is used by the authorization middleware when no policy is specified.

For more information, see dotnet/aspnetcore#9759.

Version introduced

3.0

Old behavior

Implementations of `IAuthorizationPolicyProvider` didn't require a `GetFallbackPolicyAsync` method.

New behavior

Implementations of `IAuthorizationPolicyProvider` require a `GetFallbackPolicyAsync` method.

Reason for change

A new method was needed for the new `AuthorizationMiddleware` to use when no policy is specified.

Recommended action

Add the `GetFallbackPolicyAsync` method to your implementations of `IAuthorizationPolicyProvider`.

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Authorization.IAuthorizationPolicyProvider

Caching: CompactOnMemoryPressure property removed

The ASP.NET Core 3.0 release removed the obsolete MemoryCacheOptions APIs.

Change description

This change is a follow-up to aspnet/Caching#221. For discussion, see dotnet/extensions#1062.

Version introduced

3.0

Old behavior

`MemoryCacheOptions.CompactOnMemoryPressure` property was available.

New behavior

The `MemoryCacheOptions.CompactOnMemoryPressure` property has been removed.

Reason for change

Automatically compacting the cache caused problems. To avoid unexpected behavior, the cache should only be compacted when needed.

Recommended action

To compact the cache, downcast to `MemoryCache` and call `Compact` when needed.

Category

ASP.NET Core

Affected APIs

MemoryCacheOptions.CompactOnMemoryPressure

Caching: Microsoft.Extensions.Caching.SqlServer uses new SqlClient package

The `Microsoft.Extensions.Caching.SqlServer` package will use the new `Microsoft.Data.SqlClient` package instead of `System.Data.SqlClient` package. This change could cause slight behavioral breaking changes. For more information, see Introducing the new Microsoft.Data.SqlClient.

Version introduced

3.0

Old behavior

The `Microsoft.Extensions.Caching.SqlServer` package used the `System.Data.SqlClient` package.

New behavior

`Microsoft.Extensions.Caching.SqlServer` is now using the `Microsoft.Data.SqlClient` package.

Reason for change

`Microsoft.Data.SqlClient` is a new package that is built off of `System.Data.SqlClient`. It's where all new feature work will be done from now on.

Recommended action

Customers shouldn't need to worry about this breaking change unless they were using types returned by the `Microsoft.Extensions.Caching.SqlServer` package and casting them to `System.Data.SqlClient` types. For example, if someone was casting a `DbConnection` to the old SqlConnection type, they would need to change the cast to the new `Microsoft.Data.SqlClient.SqlConnection` type.

Category

ASP.NET Core

Affected APIs

None

Caching: ResponseCaching "pubternal" types changed to internal

In ASP.NET Core 3.0, "pubternal" types in `ResponseCaching` have been changed to `internal`.

In addition, default implementations of `IResponseCachingPolicyProvider` and `IResponseCachingKeyProvider` are no longer added to services as part of the `AddResponseCaching` method.

Change description

In ASP.NET Core, "pubternal" types are declared as `public` but reside in a namespace suffixed with `.Internal`. While these types are public, they have no support policy and are subject to breaking changes. Unfortunately, accidental use of these types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Version introduced

3.0

Old behavior

These types were publicly visible, but unsupported.

New behavior

These types are now `internal`.

Reason for change

The `internal` scope better reflects the unsupported policy.

Recommended action

Copy types that are used by your app or library.

Category

ASP.NET Core

Affected APIs

- `Microsoft.AspNetCore.ResponseCaching.Internal.CachedResponse`
- `Microsoft.AspNetCore.ResponseCaching.Internal.CachedVaryByRules`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCache`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCacheEntry`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCachingKeyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.IResponseCachingPolicyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.MemoryResponseCache`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingContext`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingKeyProvider`
- `Microsoft.AspNetCore.ResponseCaching.Internal.ResponseCachingPolicyProvider`
- Microsoft.AspNetCore.ResponseCaching.ResponseCachingMiddleware.ResponseCachingMiddleware(RequestDelegate, IOptions<ResponseCachingOptions>, ILoggerFactory, IResponseCachingPolicyProvider, IResponseCache, IResponseCachingKeyProvider)

Data Protection: DataProtection.Blobs uses new Azure Storage APIs

`Azure.Extensions.AspNetCore.DataProtection.Blobs` depends on the Azure Storage libraries. These libraries renamed their assemblies, packages, and namespaces. Starting in ASP.NET Core 3.0, `Azure.Extensions.AspNetCore.DataProtection.Blobs` uses the new `Azure.Storage.`-prefixed APIs and packages.

For questions about the Azure Storage APIs, use https://github.com/Azure/azure-storage-net. For discussion on this issue, see dotnet/aspnetcore#19570.

Version introduced

3.0

Old behavior

The package referenced the `WindowsAzure.Storage` NuGet package. The package references the `Microsoft.Azure.Storage.Blob` NuGet package.

New behavior

The package references the `Azure.Storage.Blob` NuGet package.

Reason for change

This change allows `Azure.Extensions.AspNetCore.DataProtection.Blobs` to migrate to the recommended Azure Storage packages.

Recommended action

If you still need to use the older Azure Storage APIs with ASP.NET Core 3.0, add a direct dependency to the package WindowsAzure.Storage or Microsoft.Azure.Storage. This package can be installed alongside the new `Azure.Storage` APIs.

In many cases, the upgrade only involves changing the `using` statements to use the new namespaces:

diff

```
- using Microsoft.WindowsAzure.Storage;
- using Microsoft.WindowsAzure.Storage.Blob;
- using Microsoft.Azure.Storage;
- using Microsoft.Azure.Storage.Blob;
```

```
+ using Azure.Storage;
+ using Azure.Storage.Blobs;
```

Category

ASP.NET Core

Affected APIs

None

Hosting: AspNetCoreModule V1 removed from Windows Hosting Bundle

Starting with ASP.NET Core 3.0, the Windows Hosting Bundle won't contain AspNetCoreModule (ANCM) V1.

ANCM V2 is backwards compatible with ANCM OutOfProcess and is recommended for use with ASP.NET Core 3.0 apps.

For discussion, see dotnet/aspnetcore#7095 .

Version introduced

3.0

Old behavior

ANCM V1 is included in the Windows Hosting Bundle.

New behavior

ANCM V1 isn't included in the Windows Hosting Bundle.

Reason for change

ANCM V2 is backwards compatible with ANCM OutOfProcess and is recommended for use with ASP.NET Core 3.0 apps.

Recommended action

Use ANCM V2 with ASP.NET Core 3.0 apps.

If ANCM V1 is required, it can be installed using the ASP.NET Core 2.1 or 2.2 Windows Hosting Bundle.

This change will break ASP.NET Core 3.0 apps that:

- Explicitly opted into using ANCM V1 with `<AspNetCoreModuleName>AspNetCoreModule</AspNetCoreModuleName>`.
- Have a custom *web.config* file with `<add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />`.

Category

ASP.NET Core

Affected APIs

None

Hosting: Generic host restricts Startup constructor injection

The only types the generic host supports for `Startup` class constructor injection are `IHostEnvironment`, `IWebHostEnvironment`, and `IConfiguration`. Apps using `WebHost` are unaffected.

Change description

Prior to ASP.NET Core 3.0, constructor injection could be used for arbitrary types in the `Startup` class's constructor. In ASP.NET Core 3.0, the web stack was replatformed onto the generic host library. You can see the change in the *Program.cs* file of the templates:

ASP.NET Core 2.x:

https://github.com/dotnet/aspnetcore/blob/5cb615fcbe8559e49042e93394008077e30454c0/src/Templating/src/Microsoft.DotNet.Web.ProjectTemplates/content/EmptyWeb-CSharp/Program.cs#L20-L22

ASP.NET Core 3.0:

https://github.com/dotnet/aspnetcore/blob/b1ca2c1155da3920f0df5108b9fedbe82efaa11c/src/ProjectTemplates/Web.ProjectTemplates/content/EmptyWeb-CSharp/Program.cs#L19-L24

`Host` uses one dependency injection (DI) container to build the app. `WebHost` uses two containers: one for the host and one for the app. As a result, the `Startup` constructor no longer supports custom service injection. Only `IHostEnvironment`, `IWebHostEnvironment`, and `IConfiguration` can be injected. This change prevents DI issues such as the duplicate creation of a singleton service.

Version introduced

3.0

Reason for change

This change is a consequence of replatforming the web stack onto the generic host library.

Recommended action

Inject services into the `Startup.Configure` method signature. For example:

```
C#

public void Configure(IApplicationBuilder app, IOptions<MyOptions> options)
```

Category

ASP.NET Core

Affected APIs

None

Hosting: HTTPS redirection enabled for IIS out-of-process apps

Version 13.0.19218.0 of the ASP.NET Core Module (ANCM) for hosting via IIS out-of-process enables an existing HTTPS redirection feature for ASP.NET Core 3.0 and 2.2

apps.

For discussion, see dotnet/AspNetCore#15243.

Version introduced

3.0

Old behavior

The ASP.NET Core 2.1 project template first introduced support for HTTPS middleware methods like UseHttpsRedirection and UseHsts. Enabling HTTPS redirection required the addition of configuration, since apps in development don't use the default port of 443. HTTP Strict Transport Security (HSTS) is active only if the request is already using HTTPS. Localhost is skipped by default.

New behavior

In ASP.NET Core 3.0, the IIS HTTPS scenario was enhanced. With the enhancement, an app could discover the server's HTTPS ports and make `UseHttpsRedirection` work by default. The in-process component accomplished port discovery with the `IServerAddresses` feature, which only affects ASP.NET Core 3.0 apps because the in-process library is versioned with the framework. The out-of-process component changed to automatically add the `ASPNETCORE_HTTPS_PORT` environment variable. This change affected both ASP.NET Core 2.2 and 3.0 apps because the out-of-process component is shared globally. ASP.NET Core 2.1 apps aren't affected because they use a prior version of ANCM by default.

The preceding behavior was modified in ASP.NET Core 3.0.1 and 3.1.0 Preview 3 to reverse the behavior changes in ASP.NET Core 2.x. These changes only affect IIS out-of-process apps.

As detailed above, installing ASP.NET Core 3.0.0 had the side effect of also activating the `UseHttpsRedirection` middleware in ASP.NET Core 2.x apps. A change was made to ANCM in ASP.NET Core 3.0.1 and 3.1.0 Preview 3 such that installing them no longer has this effect on ASP.NET Core 2.x apps. The `ASPNETCORE_HTTPS_PORT` environment variable that ANCM populated in ASP.NET Core 3.0.0 was changed to `ASPNETCORE_ANCM_HTTPS_PORT` in ASP.NET Core 3.0.1 and 3.1.0 Preview 3. `UseHttpsRedirection` was also updated in these releases to understand both the new and old variables. ASP.NET Core 2.x won't be updated. As a result, it reverts to the previous behavior of being disabled by default.

Reason for change

Improved ASP.NET Core 3.0 functionality.

Recommended action

No action is required if you want all clients to use HTTPS. To allow some clients to use HTTP, take one of the following steps:

- Remove the calls to `UseHttpsRedirection` and `UseHsts` from your project's `Startup.Configure` method, and redeploy the app.
- In your *web.config* file, set the `ASPNETCORE_HTTPS_PORT` environment variable to an empty string. This change can occur directly on the server without redeploying the app. For example:

  XML

  ```
  <aspNetCore processPath="dotnet" arguments=".\WebApplication3.dll"
  stdoutLogEnabled="false" stdoutLogFile="\\?\%home%\LogFiles\stdout" >
      <environmentVariables>
      <environmentVariable name="ASPNETCORE_HTTPS_PORT" value="" />
      </environmentVariables>
  </aspNetCore>
  ```

`UseHttpsRedirection` can still be:

- Activated manually in ASP.NET Core 2.x by setting the `ASPNETCORE_HTTPS_PORT` environment variable to the appropriate port number (443 in most production scenarios).
- Deactivated in ASP.NET Core 3.x by defining `ASPNETCORE_ANCM_HTTPS_PORT` with an empty string value. This value is set in the same fashion as the preceding `ASPNETCORE_HTTPS_PORT` example.

Machines running ASP.NET Core 3.0.0 apps should install the ASP.NET Core 3.0.1 runtime before installing the ASP.NET Core 3.1.0 Preview 3 ANCM. Doing so ensures that `UseHttpsRedirection` continues to operate as expected for the ASP.NET Core 3.0 apps.

In Azure App Service, ANCM deploys on a separate schedule from the runtime because of its global nature. ANCM was deployed to Azure with these changes after ASP.NET Core 3.0.1 and 3.1.0 were deployed.

Category

ASP.NET Core

Affected APIs

HttpsPolicyBuilderExtensions.UseHttpsRedirection(IApplicationBuilder)

Hosting: IHostingEnvironment and IApplicationLifetime types marked obsolete and replaced

New types have been introduced to replace existing `IHostingEnvironment` and `IApplicationLifetime` types.

Version introduced

3.0

Old behavior

There were two different `IHostingEnvironment` and `IApplicationLifetime` types from `Microsoft.Extensions.Hosting` and `Microsoft.AspNetCore.Hosting`.

New behavior

The old types have been marked as obsolete and replaced with new types.

Reason for change

When `Microsoft.Extensions.Hosting` was introduced in ASP.NET Core 2.1, some types like `IHostingEnvironment` and `IApplicationLifetime` were copied from `Microsoft.AspNetCore.Hosting`. Some ASP.NET Core 3.0 changes cause apps to include both the `Microsoft.Extensions.Hosting` and `Microsoft.AspNetCore.Hosting` namespaces. Any use of those duplicate types causes an "ambiguous reference" compiler error when both namespaces are referenced.

Recommended action

Replaced any usages of the old types with the newly introduced types as below:

Obsolete types (warning):

- Microsoft.Extensions.Hosting.IHostingEnvironment
- Microsoft.AspNetCore.Hosting.IHostingEnvironment
- Microsoft.Extensions.Hosting.IApplicationLifetime
- Microsoft.AspNetCore.Hosting.IApplicationLifetime
- Microsoft.Extensions.Hosting.EnvironmentName
- Microsoft.AspNetCore.Hosting.EnvironmentName

New types:

- Microsoft.Extensions.Hosting.IHostEnvironment
- `Microsoft.AspNetCore.Hosting.IWebHostEnvironment : IHostEnvironment`
- Microsoft.Extensions.Hosting.IHostApplicationLifetime
- Microsoft.Extensions.Hosting.Environments

The new `IHostEnvironment` `IsDevelopment` and `IsProduction` extension methods are in the `Microsoft.Extensions.Hosting` namespace. That namespace may need to be added to your project.

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Hosting.EnvironmentName
- Microsoft.AspNetCore.Hosting.IApplicationLifetime
- Microsoft.AspNetCore.Hosting.IHostingEnvironment
- Microsoft.Extensions.Hosting.EnvironmentName
- Microsoft.Extensions.Hosting.IApplicationLifetime
- Microsoft.Extensions.Hosting.IHostingEnvironment

Hosting: ObjectPoolProvider removed from WebHostBuilder dependencies

As part of making ASP.NET Core more pay for play, the `ObjectPoolProvider` was removed from the main set of dependencies. Specific components relying on `ObjectPoolProvider` now add it themselves.

For discussion, see dotnet/aspnetcore#5944 .

Version introduced

3.0

Old behavior

`WebHostBuilder` provides `ObjectPoolProvider` by default in the DI container.

New behavior

`WebHostBuilder` no longer provides `ObjectPoolProvider` by default in the DI container.

Reason for change

This change was made to make ASP.NET Core more pay for play.

Recommended action

If your component requires `ObjectPoolProvider`, it needs to be added to your dependencies via the `IServiceCollection`.

Category

ASP.NET Core

Affected APIs

None

HTTP: DefaultHttpContext extensibility removed

As part of ASP.NET Core 3.0 performance improvements, the extensibility of `DefaultHttpContext` was removed. The class is now `sealed`. For more information, see dotnet/aspnetcore#6504 .

If your unit tests use `Mock<DefaultHttpContext>`, use `Mock<HttpContext>` or `new DefaultHttpContext()` instead.

For discussion, see dotnet/aspnetcore#6534 .

Version introduced

3.0

Old behavior

Classes can derive from `DefaultHttpContext`.

New behavior

Classes can't derive from `DefaultHttpContext`.

Reason for change

The extensibility was provided initially to allow pooling of the `HttpContext`, but it introduced unnecessary complexity and impeded other optimizations.

Recommended action

If you're using `Mock<DefaultHttpContext>` in your unit tests, begin using `Mock<HttpContext>` instead.

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Http.DefaultHttpContext

HTTP: HeaderNames constants changed to static readonly

Starting in ASP.NET Core 3.0 Preview 5, the fields in Microsoft.Net.Http.Headers.HeaderNames changed from `const` to `static readonly`.

For discussion, see dotnet/aspnetcore#9514 .

Version introduced

3.0

Old behavior

These fields used to be `const`.

New behavior

These fields are now `static readonly`.

Reason for change

The change:

- Prevents the values from being embedded across assembly boundaries, allowing for value corrections as needed.
- Enables faster reference equality checks.

Recommended action

Recompile against 3.0. Source code using these fields in the following ways can no longer do so:

- As an attribute argument
- As a `case` in a `switch` statement
- When defining another `const`

To work around the breaking change, switch to using self-defined header name constants or string literals.

Category

ASP.NET Core

Affected APIs

Microsoft.Net.Http.Headers.HeaderNames

HTTP: Response body infrastructure changes

The infrastructure backing an HTTP response body has changed. If you're using `HttpResponse` directly, you shouldn't need to make any code changes. Read further if you're wrapping or replacing `HttpResponse.Body` or accessing `HttpContext.Features`.

Version introduced

3.0

Old behavior

There were three APIs associated with the HTTP response body:

- `IHttpResponseFeature.Body`
- `IHttpSendFileFeature.SendFileAsync`
- `IHttpBufferingFeature.DisableResponseBuffering`

New behavior

If you replace `HttpResponse.Body`, it replaces the entire `IHttpResponseBodyFeature` with a wrapper around your given stream using `StreamResponseBodyFeature` to provide default implementations for all of the expected APIs. Setting back the original stream reverts this change.

Reason for change

The motivation is to combine the response body APIs into a single new feature interface.

Recommended action

Use `IHttpResponseBodyFeature` where you previously were using `IHttpResponseFeature.Body`, `IHttpSendFileFeature`, or `IHttpBufferingFeature`.

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Http.Features.IHttpBufferingFeature
- Microsoft.AspNetCore.Http.Features.IHttpResponseFeature.Body

- Microsoft.AspNetCore.Http.Features.IHttpSendFileFeature

HTTP: Some cookie SameSite defaults changed to None

`SameSite` is an option for cookies that can help mitigate some Cross-Site Request Forgery (CSRF) attacks. When this option was initially introduced, inconsistent defaults were used across various ASP.NET Core APIs. The inconsistency has led to confusing results. As of ASP.NET Core 3.0, these defaults are better aligned. You must opt in to this feature on a per-component basis.

Version introduced

3.0

Old behavior

Similar ASP.NET Core APIs used different default SameSiteMode values. An example of the inconsistency is seen in `HttpResponse.Cookies.Append(String, String)` and `HttpResponse.Cookies.Append(String, String, CookieOptions)`, which defaulted to `SameSiteMode.None` and `SameSiteMode.Lax`, respectively.

New behavior

All the affected APIs default to `SameSiteMode.None`.

Reason for change

The default value was changed to make `SameSite` an opt-in feature.

Recommended action

Each component that emits cookies needs to decide if `SameSite` is appropriate for its scenarios. Review your usage of the affected APIs and reconfigure `SameSite` as needed.

Category

ASP.NET Core

Affected APIs

- IResponseCookies.Append(String, String, CookieOptions)
- CookiePolicyOptions.MinimumSameSitePolicy

HTTP: Synchronous IO disabled in all servers

Starting with ASP.NET Core 3.0, synchronous server operations are disabled by default.

Change description

`AllowSynchronousIO` is an option in each server that enables or disables synchronous IO APIs like `HttpRequest.Body.Read`, `HttpResponse.Body.Write`, and `Stream.Flush`. These APIs have long been a source of thread starvation and app hangs. Starting in ASP.NET Core 3.0 Preview 3, these synchronous operations are disabled by default.

Affected servers:

- Kestrel
- HttpSys
- IIS in-process
- TestServer

Expect errors similar to:

- `Synchronous operations are disallowed. Call ReadAsync or set AllowSynchronousIO to true instead.`
- `Synchronous operations are disallowed. Call WriteAsync or set AllowSynchronousIO to true instead.`
- `Synchronous operations are disallowed. Call FlushAsync or set AllowSynchronousIO to true instead.`

Each server has an `AllowSynchronousIO` option that controls this behavior and the default for all of them is now `false`.

The behavior can also be overridden on a per-request basis as a temporary mitigation. For example:

C#

```
var syncIOFeature = HttpContext.Features.Get<IHttpBodyControlFeature>();
if (syncIOFeature != null)
{
```

```
    syncIOFeature.AllowSynchronousIO = true;
}
```

If you have trouble with a `TextWriter` or another stream calling a synchronous API in `Dispose`, call the new `DisposeAsync` API instead.

For discussion, see dotnet/aspnetcore#7644 .

Version introduced

3.0

Old behavior

`HttpRequest.Body.Read`, `HttpResponse.Body.Write`, and `Stream.Flush` were allowed by default.

New behavior

These synchronous APIs are disallowed by default:

Expect errors similar to:

- `Synchronous operations are disallowed. Call ReadAsync or set AllowSynchronousIO to true instead.`
- `Synchronous operations are disallowed. Call WriteAsync or set AllowSynchronousIO to true instead.`
- `Synchronous operations are disallowed. Call FlushAsync or set AllowSynchronousIO to true instead.`

Reason for change

These synchronous APIs have long been a source of thread starvation and app hangs. Starting in ASP.NET Core 3.0 Preview 3, the synchronous operations are disabled by default.

Recommended action

Use the asynchronous versions of the methods. The behavior can also be overridden on a per-request basis as a temporary mitigation.

C#

```
var syncIOFeature = HttpContext.Features.Get<IHttpBodyControlFeature>();
if (syncIOFeature != null)
{
    syncIOFeature.AllowSynchronousIO = true;
}
```

Category

ASP.NET Core

Affected APIs

- Stream.Flush
- Stream.Read
- Stream.Write

Identity: AddDefaultUI method overload removed

Starting with ASP.NET Core 3.0, the IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder,UIFramework) method overload no longer exists.

Version introduced

3.0

Reason for change

This change was a result of adoption of the static web assets feature.

Recommended action

Call IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder) instead of the overload that takes two arguments. If you're using Bootstrap 3, also add the following line to a `<PropertyGroup>` element in your project file:

XML

```
<IdentityUIFrameworkVersion>Bootstrap3</IdentityUIFrameworkVersion>
```

Category

ASP.NET Core

Affected APIs

IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder,UIFramework)

Identity: Default Bootstrap version of UI changed

Starting in ASP.NET Core 3.0, Identity UI defaults to using version 4 of Bootstrap.

Version introduced

3.0

Old behavior

The `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();` method call was the same as `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI(UIFramework.Bootstrap3);`

New behavior

The `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();` method call is the same as `services.AddDefaultIdentity<IdentityUser>().AddDefaultUI(UIFramework.Bootstrap4);`

Reason for change

Bootstrap 4 was released during ASP.NET Core 3.0 timeframe.

Recommended action

You're impacted by this change if you use the default Identity UI and have added it in `Startup.ConfigureServices` as shown in the following example:

```
C#

services.AddDefaultIdentity<IdentityUser>().AddDefaultUI();
```

Take one of the following actions:

- Migrate your app to use Bootstrap 4 using their migration guide.
- Update `Startup.ConfigureServices` to enforce usage of Bootstrap 3. For example:

```
C#

services.AddDefaultIdentity<IdentityUser>
().AddDefaultUI(UIFramework.Bootstrap3);
```

Category

ASP.NET Core

Affected APIs

None

Identity: SignInAsync throws exception for unauthenticated identity

By default, `SignInAsync` throws an exception for principals / identities in which `IsAuthenticated` is `false`.

Version introduced

3.0

Old behavior

`SignInAsync` accepts any principals / identities, including identities in which `IsAuthenticated` is `false`.

New behavior

By default, `SignInAsync` throws an exception for principals / identities in which `IsAuthenticated` is `false`. There's a new flag to suppress this behavior, but the default behavior has changed.

Reason for change

The old behavior was problematic because, by default, these principals were rejected by `[Authorize]` / `RequireAuthenticatedUser()`.

Recommended action

In ASP.NET Core 3.0 Preview 6, there's a `RequireAuthenticatedSignIn` flag on `AuthenticationOptions` that is `true` by default. Set this flag to `false` to restore the old behavior.

Category

ASP.NET Core

Affected APIs

None

Identity: SignInManager constructor accepts new parameter

Starting with ASP.NET Core 3.0, a new `IUserConfirmation<TUser>` parameter was added to the `SignInManager` constructor. For more information, see dotnet/aspnetcore#8356.

Version introduced

3.0

Reason for change

The motivation for the change was to add support for new email / confirmation flows in Identity.

Recommended action

If manually constructing a `SignInManager`, provide an implementation of `IUserConfirmation` or grab one from dependency injection to provide.

Category

ASP.NET Core

Affected APIs

SignInManager<TUser>

Identity: UI uses static web assets feature

ASP.NET Core 3.0 introduced a static web assets feature, and Identity UI has adopted it.

Change description

As a result of Identity UI adopting the static web assets feature:

- Framework selection is accomplished by using the `IdentityUIFrameworkVersion` property in your project file.
- Bootstrap 4 is the default UI framework for Identity UI. Bootstrap 3 has reached end of life, and you should consider migrating to a supported version.

Version introduced

3.0

Old behavior

The default UI framework for Identity UI was **Bootstrap 3**. The UI framework could be configured using a parameter to the `AddDefaultUI` method call in `Startup.ConfigureServices`.

New behavior

The default UI framework for Identity UI is **Bootstrap 4**. The UI framework must be configured in your project file, instead of in the `AddDefaultUI` method call.

Reason for change

Adoption of the static web assets feature required that the UI framework configuration move to MSBuild. The decision on which framework to embed is a build-time decision, not a runtime decision.

Recommended action

Review your site UI to ensure the new Bootstrap 4 components are compatible. If necessary, use the `IdentityUIFrameworkVersion` MSBuild property to revert to Bootstrap 3. Add the property to a `<PropertyGroup>` element in your project file:

```
XML

<IdentityUIFrameworkVersion>Bootstrap3</IdentityUIFrameworkVersion>
```

Category

ASP.NET Core

Affected APIs

IdentityBuilderUIExtensions.AddDefaultUI(IdentityBuilder, UIFramework)

Kestrel: Connection adapters removed

As part of the move to move "pubternal" APIs to `public`, the concept of an `IConnectionAdapter` was removed from Kestrel. Connection adapters are being replaced with connection middleware (similar to HTTP middleware in the ASP.NET Core pipeline, but for lower-level connections). HTTPS and connection logging have moved from connection adapters to connection middleware. Those extension methods should continue to work seamlessly, but the implementation details have changed.

For more information, see dotnet/aspnetcore#11412. For discussion, see dotnet/aspnetcore#11475.

Version introduced

3.0

Old behavior

Kestrel extensibility components were created using `IConnectionAdapter`.

New behavior

Kestrel extensibility components are created as middleware.

Reason for change

This change is intended to provide a more flexible extensibility architecture.

Recommended action

Convert any implementations of `IConnectionAdapter` to use the new middleware pattern as shown here.

Category

ASP.NET Core

Affected APIs

`Microsoft.AspNetCore.Server.Kestrel.Core.Adapter.Internal.IConnectionAdapter`

Kestrel: Empty HTTPS assembly removed

The assembly Microsoft.AspNetCore.Server.Kestrel.Https has been removed.

Version introduced

3.0

Reason for change

In ASP.NET Core 2.1, the contents of `Microsoft.AspNetCore.Server.Kestrel.Https` were moved to Microsoft.AspNetCore.Server.Kestrel.Core. This change was done in a non-breaking way using `[TypeForwardedTo]` attributes.

Recommended action

- Libraries referencing `Microsoft.AspNetCore.Server.Kestrel.Https` 2.0 should update all ASP.NET Core dependencies to 2.1 or later. Otherwise, they may break when loaded into an ASP.NET Core 3.0 app.
- Apps and libraries targeting ASP.NET Core 2.1 and later should remove any direct references to the `Microsoft.AspNetCore.Server.Kestrel.Https` NuGet package.

Category

ASP.NET Core

Affected APIs

None

Kestrel: Request trailer headers moved to new collection

In prior versions, Kestrel added HTTP/1.1 chunked trailer headers into the request headers collection when the request body was read to the end. This behavior caused concerns about ambiguity between headers and trailers. The decision was made to move the trailers to a new collection.

HTTP/2 request trailers were unavailable in ASP.NET Core 2.2 but are now also available in this new collection in ASP.NET Core 3.0.

New request extension methods have been added to access these trailers.

HTTP/1.1 trailers are available once the entire request body has been read.

HTTP/2 trailers are available once they're received from the client. The client won't send the trailers until the entire request body has been at least buffered by the server. You may need to read the request body to free up buffer space. Trailers are always available if you read the request body to the end. The trailers mark the end of the body.

Version introduced

3.0

Old behavior

Request trailer headers would be added to the `HttpRequest.Headers` collection.

New behavior

Request trailer headers **aren't present** in the `HttpRequest.Headers` collection. Use the following extension methods on `HttpRequest` to access them:

- `GetDeclaredTrailers()` - Gets the request "Trailer" header that lists which trailers to expect after the body.
- `SupportsTrailers()` - Indicates if the request supports receiving trailer headers.
- `CheckTrailersAvailable()` - Determines if the request supports trailers and if they're available for reading.
- `GetTrailer(string trailerName)` - Gets the requested trailing header from the response.

Reason for change

Trailers are a key feature in scenarios like gRPC. Merging the trailers in to request headers was confusing to users.

Recommended action

Use the trailer-related extension methods on `HttpRequest` to access trailers.

Category

ASP.NET Core

Affected APIs

HttpRequest.Headers

Kestrel: Transport abstractions removed and made public

As part of moving away from "pubternal" APIs, the Kestrel transport layer APIs are exposed as a public interface in the `Microsoft.AspNetCore.Connections.Abstractions` library.

Version introduced

3.0

Old behavior

- Transport-related abstractions were available in the `Microsoft.AspNetCore.Server.Kestrel.Transport.Abstractions` library.
- The `ListenOptions.NoDelay` property was available.

New behavior

- The `IConnectionListener` interface was introduced in the `Microsoft.AspNetCore.Connections.Abstractions` library to expose the most used functionality from the `...Transport.Abstractions` library.
- The `NoDelay` is now available in transport options (`LibuvTransportOptions` and `SocketTransportOptions`).
- `SchedulingMode` is no longer available.

Reason for change

ASP.NET Core 3.0 has moved away from "pubternal" APIs.

Recommended action

Category

ASP.NET Core

Affected APIs

None

Localization: ResourceManagerWithCultureStringLocalizer and WithCulture marked obsolete

The ResourceManagerWithCultureStringLocalizer class and WithCulture interface member are often sources of confusion for users of localization, especially when creating their own `IStringLocalizer` implementation. These items give the user the impression that an `IStringLocalizer` instance is "per-language, per-resource". In reality, the instances should only be "per-resource". The language searched for is determined by the `CultureInfo.CurrentUICulture` at execution time. To eliminate the source of

confusion, the APIs were marked as obsolete in ASP.NET Core 3.0 Preview 3. The APIs will be removed in a future release.

For context, see dotnet/aspnetcore#3324. For discussion, see dotnet/aspnetcore#7756.

Version introduced

3.0

Old behavior

Methods weren't marked as `Obsolete`.

New behavior

Methods are marked `Obsolete`.

Reason for change

The APIs represented a use case that isn't recommended. There was confusion about the design of localization.

Recommended action

The recommendation is to use `ResourceManagerStringLocalizer` instead. Let the culture be set by the `CurrentCulture`. If that isn't an option, create and use a copy of ResourceManagerWithCultureStringLocalizer.

Category

ASP.NET Core

Affected APIs

- ResourceManagerWithCultureStringLocalizer
- ResourceManagerStringLocalizer.WithCulture

Logging: DebugLogger class made internal

Prior to ASP.NET Core 3.0, `DebugLogger`'s access modifier was `public`. In ASP.NET Core 3.0, the access modifier changed to `internal`.

Version introduced

3.0

Reason for change

The change is being made to:

- Enforce consistency with other logger implementations such as `ConsoleLogger`.
- Reduce the API surface.

Recommended action

Use the AddDebug `ILoggingBuilder` extension method to enable debug logging. DebugLoggerProvider is also still `public` in the event the service needs to be registered manually.

Category

ASP.NET Core

Affected APIs

Microsoft.Extensions.Logging.Debug.DebugLogger

MVC: Async suffix trimmed from controller action names

As part of addressing dotnet/aspnetcore#4849, ASP.NET Core MVC trims the suffix `Async` from action names by default. Starting with ASP.NET Core 3.0, this change affects both routing and link generation.

Version introduced

3.0

Old behavior

Consider the following ASP.NET Core MVC controller:

C#

```
public class ProductController : Controller
{
    public async IActionResult ListAsync()
    {
        var model = await DbContext.Products.ToListAsync();
        return View(model);
    }
}
```

The action is routable via `Product/ListAsync`. Link generation requires specifying the `Async` suffix. For example:

CSHTML

```
<a asp-controller="Product" asp-action="ListAsync">List</a>
```

New behavior

In ASP.NET Core 3.0, the action is routable via `Product/List`. Link generation code should omit the `Async` suffix. For example:

CSHTML

```
<a asp-controller="Product" asp-action="List">List</a>
```

This change doesn't affect names specified using the `[ActionName]` attribute. The new behavior can be disabled by setting `MvcOptions.SuppressAsyncSuffixInActionNames` to `false` in `Startup.ConfigureServices`:

C#

```
services.AddMvc(options =>
{
   options.SuppressAsyncSuffixInActionNames = false;
});
```

Reason for change

By convention, asynchronous .NET methods are suffixed with `Async`. However, when a method defines an MVC action, it's undesirable to use the `Async` suffix.

Recommended action

If your app depends on MVC actions preserving the name's `Async` suffix, choose one of the following mitigations:

- Use the `[ActionName]` attribute to preserve the original name.
- Disable the renaming entirely by setting `MvcOptions.SuppressAsyncSuffixInActionNames` to `false` in `Startup.ConfigureServices`:

C#

```
services.AddMvc(options =>
{
   options.SuppressAsyncSuffixInActionNames = false;
});
```

Category

ASP.NET Core

Affected APIs

None

MVC: JsonResult moved to Microsoft.AspNetCore.Mvc.Core

`JsonResult` has moved to the `Microsoft.AspNetCore.Mvc.Core` assembly. This type used to be defined in Microsoft.AspNetCore.Mvc.Formatters.Json. An assembly-level [TypeForwardedTo] attribute was added to `Microsoft.AspNetCore.Mvc.Formatters.Json` to address this issue for the majority of users. Apps that use third-party libraries may encounter issues.

Version introduced

3.0 Preview 6

Old behavior

An app using a 2.2-based library builds successfully.

New behavior

An app using a 2.2-based library fails compilation. An error containing a variation of the following text is provided:

Output

```
The type 'JsonResult' exists in both 'Microsoft.AspNetCore.Mvc.Core,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=adb9793829ddae60' and
'Microsoft.AspNetCore.Mvc.Formatters.Json, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=adb9793829ddae60'
```

For an example of such an issue, see dotnet/aspnetcore#7220.

Reason for change

Platform-level changes to the composition of ASP.NET Core as described at aspnet/Announcements#325.

Recommended action

Libraries compiled against the 2.2 version of `Microsoft.AspNetCore.Mvc.Formatters.Json` may need to recompile to address the problem for all consumers. If affected, contact the library author. Request recompilation of the library to target ASP.NET Core 3.0.

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Mvc.JsonResult

MVC: Precompilation tool deprecated

In ASP.NET Core 1.1, the `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` (MVC precompilation tool) package was introduced to add support for publish-time

compilation of Razor files (*.cshtml* files). In ASP.NET Core 2.1, the Razor SDK was introduced to expand upon features of the precompilation tool. The Razor SDK added support for build- and publish-time compilation of Razor files. The SDK verifies the correctness of *.cshtml* files at build time while improving on app startup time. The Razor SDK is on by default, and no gesture is required to start using it.

In ASP.NET Core 3.0, the ASP.NET Core 1.1-era MVC precompilation tool was removed. Earlier package versions will continue receiving important bug and security fixes in the patch release.

Version introduced

3.0

Old behavior

The `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` package was used to pre-compile MVC Razor views.

New behavior

The Razor SDK natively supports this functionality. The `Microsoft.AspNetCore.Mvc.Razor.ViewCompilation` package is no longer updated.

Reason for change

The Razor SDK provides more functionality and verifies the correctness of *.cshtml* files at build time. The SDK also improves app startup time.

Recommended action

For users of ASP.NET Core 2.1 or later, update to use the native support for precompilation in the Razor SDK. If bugs or missing features prevent migration to the Razor SDK, open an issue at dotnet/aspnetcore.

Category

ASP.NET Core

Affected APIs

None

MVC: "Pubternal" types changed to internal

In ASP.NET Core 3.0, all "pubternal" types in MVC were updated to either be `public` in a supported namespace or `internal` as appropriate.

Change description

In ASP.NET Core, "pubternal" types are declared as `public` but reside in a `.Internal`-suffixed namespace. While these types are `public`, they have no support policy and are subject to breaking changes. Unfortunately, accidental use of these types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Version introduced

3.0

Old behavior

Some types in MVC were `public` but in a `.Internal` namespace. These types had no support policy and were subject to breaking changes.

New behavior

All such types are updated either to be `public` in a supported namespace or marked as `internal`.

Reason for change

Accidental use of the "pubternal" types has been common, resulting in breaking changes to these projects and limiting the ability to maintain the framework.

Recommended action

If you're using types that have become truly `public` and have been moved into a new, supported namespace, update your references to match the new namespaces.

If you're using types that have become marked as `internal`, you'll need to find an alternative. The previously "pubternal" types were never supported for public use. If there are specific types in these namespaces that are critical to your apps, file an issue at dotnet/aspnetcore. Considerations may be made for making the requested types `public`.

Category

ASP.NET Core

Affected APIs

This change includes types in the following namespaces:

- `Microsoft.AspNetCore.Mvc.Cors.Internal`
- `Microsoft.AspNetCore.Mvc.DataAnnotations.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Json.Internal`
- `Microsoft.AspNetCore.Mvc.Formatters.Xml.Internal`
- `Microsoft.AspNetCore.Mvc.Internal`
- `Microsoft.AspNetCore.Mvc.ModelBinding.Internal`
- `Microsoft.AspNetCore.Mvc.Razor.Internal`
- `Microsoft.AspNetCore.Mvc.RazorPages.Internal`
- `Microsoft.AspNetCore.Mvc.TagHelpers.Internal`
- `Microsoft.AspNetCore.Mvc.ViewFeatures.Internal`

MVC: Web API compatibility shim removed

Starting with ASP.NET Core 3.0, the `Microsoft.AspNetCore.Mvc.WebApiCompatShim` package is no longer available.

Change description

The `Microsoft.AspNetCore.Mvc.WebApiCompatShim` (WebApiCompatShim) package provides partial compatibility in ASP.NET Core with ASP.NET 4.x Web API 2 to simplify migrating existing Web API implementations to ASP.NET Core. However, apps using the WebApiCompatShim don't benefit from the API-related features shipping in recent ASP.NET Core releases. Such features include improved Open API specification generation, standardized error handling, and client code generation. To better focus the

API efforts in 3.0, WebApiCompatShim was removed. Existing apps using the WebApiCompatShim should migrate to the newer `[ApiController]` model.

Version introduced

3.0

Reason for change

The Web API compatibility shim was a migration tool. It restricts user access to new functionality added in ASP.NET Core.

Recommended action

Remove usage of this shim and migrate directly to the similar functionality in ASP.NET Core itself.

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Mvc.WebApiCompatShim

Razor: RazorTemplateEngine API removed

The `RazorTemplateEngine` API was removed and replaced with `Microsoft.AspNetCore.Razor.Language.RazorProjectEngine`.

For discussion, see GitHub issue dotnet/aspnetcore#25215.

Version introduced

3.0

Old behavior

A template engine can be created and used to parse and generate code for Razor files.

New behavior

`RazorProjectEngine` can be created and provided the same type of information as `RazorTemplateEngine` to parse and generate code for Razor files. `RazorProjectEngine` also provides extra levels of configuration.

Reason for change

`RazorTemplateEngine` was too tightly coupled to the existing implementations. This tight coupling led to more questions when trying to properly configure a Razor parsing/generation pipeline.

Recommended action

Use `RazorProjectEngine` instead of `RazorTemplateEngine`. Consider the following examples.

Create and configure the RazorProjectEngine

C#

```
RazorProjectEngine projectEngine =
    RazorProjectEngine.Create(RazorConfiguration.Default,

RazorProjectFileSystem.Create(@"C:\source\repos\ConsoleApp4\ConsoleApp4"),
        builder =>
        {
            builder.ConfigureClass((document, classNode) =>
            {
                classNode.ClassName = "MyClassName";

                // Can also configure other aspects of the class here.
            });

            // More configuration can go here
        });
```

Generate code for a Razor file

C#

```
RazorProjectItem item = projectEngine.FileSystem.GetItem(
    @"C:\source\repos\ConsoleApp4\ConsoleApp4\Example.cshtml",
    FileKinds.Legacy);
RazorCodeDocument output = projectEngine.Process(item);
```

```
// Things available
RazorSyntaxTree syntaxTree = output.GetSyntaxTree();
DocumentIntermediateNode intermediateDocument =
    output.GetDocumentIntermediateNode();
RazorCSharpDocument csharpDocument = output.GetCSharpDocument();
```

Category

ASP.NET Core

Affected APIs

- `RazorTemplateEngine`
- `RazorTemplateEngineOptions`

Razor: Runtime compilation moved to a package

Support for runtime compilation of Razor views and Razor Pages has moved to a separate package.

Version introduced

3.0

Old behavior

Runtime compilation is available without needing additional packages.

New behavior

The functionality has been moved to the Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation package.

The following APIs were previously available in `Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions` to support runtime compilation. The APIs are now available via `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation.MvcRazorRuntimeCompilationOptions`.

- `RazorViewEngineOptions.FileProviders` is now `MvcRazorRuntimeCompilationOptions.FileProviders`
- `RazorViewEngineOptions.AdditionalCompilationReferences` is now `MvcRazorRuntimeCompilationOptions.AdditionalReferencePaths`

In addition, `Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions.AllowRecompilingViewsOnFileChange` has been removed. Recompilation on file changes is enabled by default by referencing the `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation` package.

Reason for change

This change was necessary to remove the ASP.NET Core shared framework dependency on Roslyn.

Recommended action

Apps that require runtime compilation or recompilation of Razor files should take the following steps:

1. Add a reference to the `Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation` package.
2. Update the project's `Startup.ConfigureServices` method to include a call to `AddRazorRuntimeCompilation`. For example:

```
C#

services.AddMvc()
    .AddRazorRuntimeCompilation();
```

Category

ASP.NET Core

Affected APIs

Microsoft.AspNetCore.Mvc.Razor.RazorViewEngineOptions

Session state: Obsolete APIs removed

Obsolete APIs for configuring session cookies were removed. For more information, see aspnet/Announcements#257.

Version introduced

3.0

Reason for change

This change enforces consistency across APIs for configuring features that use cookies.

Recommended action

Migrate usage of the removed APIs to their newer replacements. Consider the following example in `Startup.ConfigureServices`:

C#

```
public void ConfigureServices(ServiceCollection services)
{
    services.AddSession(options =>
    {
        // Removed obsolete APIs
        options.CookieName = "SessionCookie";
        options.CookieDomain = "contoso.com";
        options.CookiePath = "/";
        options.CookieHttpOnly = true;
        options.CookieSecure = CookieSecurePolicy.Always;

        // new API
        options.Cookie.Name = "SessionCookie";
        options.Cookie.Domain = "contoso.com";
        options.Cookie.Path = "/";
        options.Cookie.HttpOnly = true;
        options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
    });
}
```

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Builder.SessionOptions.CookieDomain

- Microsoft.AspNetCore.Builder.SessionOptions.CookieHttpOnly
- Microsoft.AspNetCore.Builder.SessionOptions.CookieName
- Microsoft.AspNetCore.Builder.SessionOptions.CookiePath
- Microsoft.AspNetCore.Builder.SessionOptions.CookieSecure

Shared framework: Assemblies removed from Microsoft.AspNetCore.App

Starting in ASP.NET Core 3.0, the ASP.NET Core shared framework (`Microsoft.AspNetCore.App`) only contains first-party assemblies that are fully developed, supported, and serviceable by Microsoft.

Change description

Think of the change as the redefining of boundaries for the ASP.NET Core "platform." The shared framework will be source-buildable by anybody via GitHub and will continue to offer the existing benefits of .NET Core shared frameworks to your apps. Some benefits include smaller deployment size, centralized patching, and faster startup time.

As part of the change, some notable breaking changes are introduced in `Microsoft.AspNetCore.App`.

Version introduced

3.0

Old behavior

Projects referenced `Microsoft.AspNetCore.App` via a `<PackageReference>` element in the project file.

Additionally, `Microsoft.AspNetCore.App` contained the following subcomponents:

- Json.NET (`Newtonsoft.Json`)
- Entity Framework Core (assemblies prefixed with `Microsoft.EntityFrameworkCore.`)
- Roslyn (`Microsoft.CodeAnalysis`)

New behavior

A reference to `Microsoft.AspNetCore.App` no longer requires a `<PackageReference>` element in the project file. The .NET Core SDK supports a new element called `<FrameworkReference>`, which replaces the use of `<PackageReference>`.

For more information, see dotnet/aspnetcore#3612.

Entity Framework Core ships as NuGet packages. This change aligns the shipping model with all other data access libraries on .NET. It provides Entity Framework Core the simplest path to continue innovating while supporting the various .NET platforms. The move of Entity Framework Core out of the shared framework has no impact on its status as a Microsoft-developed, supported, and serviceable library. The .NET Core support policy continues to cover it.

Json.NET and Entity Framework Core continue to work with ASP.NET Core. They won't, however, be included in the shared framework.

For more information, see The future of JSON in .NET Core 3.0. Also see the complete list of binaries removed from the shared framework.

Reason for change

This change simplifies the consumption of `Microsoft.AspNetCore.App` and reduces the duplication between NuGet packages and shared frameworks.

For more information on the motivation for this change, see this blog post.

Recommended action

Starting with ASP.NET Core 3.0, it is no longer necessary for projects to consume assemblies in `Microsoft.AspNetCore.App` as NuGet packages. To simplify the targeting and usage of the ASP.NET Core shared framework, many NuGet packages shipped since ASP.NET Core 1.0 are no longer produced. The APIs those packages provide are still available to apps by using a `<FrameworkReference>` to `Microsoft.AspNetCore.App`. Common API examples include Kestrel, MVC, and Razor.

This change doesn't apply to all binaries referenced via `Microsoft.AspNetCore.App` in ASP.NET Core 2.x. Notable exceptions include:

- `Microsoft.Extensions` libraries that continue to target .NET Standard are available as NuGet packages (see https://github.com/dotnet/extensions).
- APIs produced by the ASP.NET Core team that aren't part of `Microsoft.AspNetCore.App`. For example, the following components are available as

NuGet packages:
    - Entity Framework Core
    - APIs that provide third-party integration
    - Experimental features
    - APIs with dependencies that couldn't fulfill the requirements to be in the shared framework
- Extensions to MVC that maintain support for Json.NET. An API is provided as a NuGet package to support using Json.NET and MVC. See the ASP.NET Core migration guide for more details.
- The SignalR .NET client continues to support .NET Standard and ships as a NuGet package. It's intended for use on many .NET runtimes, such as Xamarin and UWP.

For more information, see Stop producing packages for shared framework assemblies in 3.0. For discussion, see dotnet/aspnetcore#3757.

Category

ASP.NET Core

Affected APIs

- Microsoft.CodeAnalysis
- Microsoft.EntityFrameworkCore

Shared framework: Removed Microsoft.AspNetCore.All

Starting in ASP.NET Core 3.0, the `Microsoft.AspNetCore.All` metapackage and the matching `Microsoft.AspNetCore.All` shared framework are no longer produced. This package is available in ASP.NET Core 2.2 and will continue to receive servicing updates in ASP.NET Core 2.1.

Version introduced

3.0

Old behavior

Apps could use the `Microsoft.AspNetCore.All` metapackage to target the `Microsoft.AspNetCore.All` shared framework on .NET Core.

New behavior

.NET Core 3.0 doesn't include a `Microsoft.AspNetCore.All` shared framework.

Reason for change

The `Microsoft.AspNetCore.All` metapackage included a large number of external dependencies.

Recommended action

Migrate your project to use the `Microsoft.AspNetCore.App` framework. Components that were previously available in `Microsoft.AspNetCore.All` are still available on NuGet. Those components are now deployed with your app instead of being included in the shared framework.

Category

ASP.NET Core

Affected APIs

None

SignalR: HandshakeProtocol.SuccessHandshakeData replaced

The HandshakeProtocol.SuccessHandshakeData field was removed and replaced with a helper method that generates a successful handshake response given a specific `IHubProtocol`.

Version introduced

3.0

Old behavior

`HandshakeProtocol.SuccessHandshakeData` was a `public static ReadOnlyMemory<byte>` field.

New behavior

`HandshakeProtocol.SuccessHandshakeData` has been replaced by a `static GetSuccessfulHandshake(IHubProtocol protocol)` method that returns a `ReadOnlyMemory<byte>` based on the specified protocol.

Reason for change

Additional fields were added to the handshake *response* that are non-constant and change depending on the selected protocol.

Recommended action

None. This type isn't designed for use from user code. It's `public`, so it can be shared between the SignalR server and client. It may also be used by customer SignalR clients written in .NET. **Users** of SignalR shouldn't be affected by this change.

Category

ASP.NET Core

Affected APIs

HandshakeProtocol.SuccessHandshakeData

SignalR: HubConnection ResetSendPing and ResetTimeout methods removed

The `ResetSendPing` and `ResetTimeout` methods were removed from the SignalR `HubConnection` API. These methods were originally intended only for internal use but were made public in ASP.NET Core 2.2. These methods won't be available starting in the ASP.NET Core 3.0 Preview 4 release. For discussion, see dotnet/aspnetcore#8543.

Version introduced

3.0

Old behavior

APIs were available.

New behavior

APIs are removed.

Reason for change

These methods were originally intended only for internal use but were made public in ASP.NET Core 2.2.

Recommended action

Don't use these methods.

Category

ASP.NET Core

Affected APIs

- HubConnection.ResetSendPing()
- HubConnection.ResetTimeout()

SignalR: HubConnectionContext constructors changed

SignalR's `HubConnectionContext` constructors changed to accept an options type, rather than multiple parameters, to future-proof adding options. This change replaces two constructors with a single constructor that accepts an options type.

Version introduced

3.0

Old behavior

`HubConnectionContext` has two constructors:

C#

```
public HubConnectionContext(ConnectionContext connectionContext, TimeSpan
keepAliveInterval, ILoggerFactory loggerFactory);
public HubConnectionContext(ConnectionContext connectionContext, TimeSpan
```

```
keepAliveInterval, ILoggerFactory loggerFactory, TimeSpan
clientTimeoutInterval);
```

New behavior

The two constructors were removed and replaced with one constructor:

C#

```
public HubConnectionContext(ConnectionContext connectionContext,
HubConnectionContextOptions contextOptions, ILoggerFactory loggerFactory)
```

Reason for change

The new constructor uses a new options object. Consequently, the features of `HubConnectionContext` can be expanded in the future without making more constructors and breaking changes.

Recommended action

Instead of using the following constructor:

C#

```
HubConnectionContext connectionContext = new HubConnectionContext(
    connectionContext,
    keepAliveInterval: TimeSpan.FromSeconds(15),
    loggerFactory,
    clientTimeoutInterval: TimeSpan.FromSeconds(15));
```

Use the following constructor:

C#

```
HubConnectionContextOptions contextOptions = new
HubConnectionContextOptions()
{
    KeepAliveInterval = TimeSpan.FromSeconds(15),
    ClientTimeoutInterval = TimeSpan.FromSeconds(15)
};
HubConnectionContext connectionContext = new
HubConnectionContext(connectionContext, contextOptions, loggerFactory);
```

Category

ASP.NET Core

Affected APIs

- HubConnectionContext(ConnectionContext, TimeSpan, ILoggerFactory)
- HubConnectionContext(ConnectionContext, TimeSpan, ILoggerFactory, TimeSpan)

SignalR: JavaScript client package name changed

In ASP.NET Core 3.0 Preview 7, the SignalR JavaScript client package name changed from `@aspnet/signalr` to `@microsoft/signalr`. The name change reflects the fact that SignalR is useful in more than just ASP.NET Core apps, thanks to the Azure SignalR Service.

To react to this change, change references in your *package.json* files, `require` statements, and ECMAScript `import` statements. No API will change as part of this rename.

For discussion, see dotnet/aspnetcore#11637.

Version introduced

3.0

Old behavior

The client package was named `@aspnet/signalr`.

New behavior

The client package is named `@microsoft/signalr`.

Reason for change

The name change clarifies that SignalR is useful beyond ASP.NET Core apps, thanks to the Azure SignalR Service.

Recommended action

Switch to the new package `@microsoft/signalr`.

Category

ASP.NET Core

Affected APIs

None

SignalR: UseSignalR and UseConnections methods marked obsolete

The methods `UseConnections` and `UseSignalR` and the classes `ConnectionsRouteBuilder` and `HubRouteBuilder` are marked as obsolete in ASP.NET Core 3.0.

Version introduced

3.0

Old behavior

SignalR hub routing was configured using `UseSignalR` or `UseConnections`.

New behavior

The old way of configuring routing has been obsoleted and replaced with endpoint routing.

Reason for change

Middleware is being moved to the new endpoint routing system. The old way of adding middleware is being obsoleted.

Recommended action

Replace `UseSignalR` with `UseEndpoints`:

Old code:

C#

```
app.UseSignalR(routes =>
{
    routes.MapHub<SomeHub>("/path");
});
```

New code:

C#

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapHub<SomeHub>("/path");
});
```

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Builder.ConnectionsAppBuilderExtensions.UseConnections(IApplicationBuilder, Action<ConnectionsRouteBuilder>)
- Microsoft.AspNetCore.Builder.SignalRAppBuilderExtensions.UseSignalR(IApplicationBuilder, Action<HubRouteBuilder>)
- Microsoft.AspNetCore.Http.Connections.ConnectionsRouteBuilder
- Microsoft.AspNetCore.SignalR.HubRouteBuilder

SPAs: SpaServices and NodeServices marked obsolete

The contents of the following NuGet packages have all been unnecessary since ASP.NET Core 2.1. Consequently, the following packages are being marked as obsolete:

- Microsoft.AspNetCore.SpaServices
- Microsoft.AspNetCore.NodeServices

For the same reason, the following npm modules are being marked as deprecated:

- aspnet-angular
- aspnet-prerendering
- aspnet-webpack

- aspnet-webpack-react
- domain-task

The preceding packages and npm modules will later be removed in .NET 5.

Version introduced

3.0

Old behavior

The deprecated packages and npm modules were intended to integrate ASP.NET Core with various Single-Page App (SPA) frameworks. Such frameworks include Angular, React, and React with Redux.

New behavior

A new integration mechanism exists in the Microsoft.AspNetCore.SpaServices.Extensions NuGet package. The package remains the basis of the Angular and React project templates since ASP.NET Core 2.1.

Reason for change

ASP.NET Core supports integration with various Single-Page App (SPA) frameworks, including Angular, React, and React with Redux. Initially, integration with these frameworks was accomplished with ASP.NET Core-specific components that handled scenarios like server-side prerendering and integration with Webpack. As time went on, industry standards changed. Each of the SPA frameworks released their own standard command-line interfaces. For example, Angular CLI and create-react-app.

When ASP.NET Core 2.1 was released in May 2018, the team responded to the change in standards. A newer and simpler way to integrate with the SPA frameworks' own toolchains was provided. The new integration mechanism exists in the package `Microsoft.AspNetCore.SpaServices.Extensions` and remains the basis of the Angular and React project templates since ASP.NET Core 2.1.

To clarify that the older ASP.NET Core-specific components are irrelevant and not recommended:

- The pre-2.1 integration mechanism is marked as obsolete.
- The supporting npm packages are marked as deprecated.

Recommended action

If you're using these packages, update your apps to use the functionality:

- In the `Microsoft.AspNetCore.SpaServices.Extensions` package.
- Provided by the SPA frameworks you're using

To enable features like server-side prerendering and hot module reload, see the documentation for the corresponding SPA framework. The functionality in `Microsoft.AspNetCore.SpaServices.Extensions` is *not* obsolete and will continue to be supported.

Category

ASP.NET Core

Affected APIs

- Microsoft.AspNetCore.Builder.SpaRouteExtensions
- Microsoft.AspNetCore.Builder.WebpackDevMiddleware
- Microsoft.AspNetCore.NodeServices.EmbeddedResourceReader
- Microsoft.AspNetCore.NodeServices.INodeServices
- Microsoft.AspNetCore.NodeServices.NodeServicesFactory
- Microsoft.AspNetCore.NodeServices.NodeServicesOptions
- Microsoft.AspNetCore.NodeServices.StringAsTempFile
- Microsoft.AspNetCore.NodeServices.HostingModels.INodeInstance
- Microsoft.AspNetCore.NodeServices.HostingModels.NodeInvocationException
- Microsoft.AspNetCore.NodeServices.HostingModels.NodeInvocationInfo
- Microsoft.AspNetCore.NodeServices.HostingModels.NodeServicesOptionsExtensions
- Microsoft.AspNetCore.NodeServices.HostingModels.OutOfProcessNodeInstance
- Microsoft.AspNetCore.SpaServices.Prerendering.ISpaPrerenderer
- Microsoft.AspNetCore.SpaServices.Prerendering.ISpaPrerendererBuilder

- Microsoft.AspNetCore.SpaServices.Prerendering.JavaScriptModuleExport
- Microsoft.AspNetCore.SpaServices.Prerendering.Prerenderer
- Microsoft.AspNetCore.SpaServices.Prerendering.PrerenderTagHelper
- Microsoft.AspNetCore.SpaServices.Prerendering.RenderToStringResult
- Microsoft.AspNetCore.SpaServices.Webpack.WebpackDevMiddlewareOptions
- Microsoft.Extensions.DependencyInjection.NodeServicesServiceCollectionExtensions
- Microsoft.Extensions.DependencyInjection.PrerenderingServiceCollectionExtensions

---

SPAs: SpaServices and NodeServices no longer fall back to console logger

Microsoft.AspNetCore.SpaServices and Microsoft.AspNetCore.NodeServices won't display console logs unless logging is configured.

Version introduced

3.0

Old behavior

`Microsoft.AspNetCore.SpaServices` and `Microsoft.AspNetCore.NodeServices` used to automatically create a console logger when logging isn't configured.

New behavior

`Microsoft.AspNetCore.SpaServices` and `Microsoft.AspNetCore.NodeServices` won't display console logs unless logging is configured.

Reason for change

There's a need to align with how other ASP.NET Core packages implement logging.

Recommended action

If the old behavior is required, to configure console logging, add `services.AddLogging(builder => builder.AddConsole())` to your `Setup.ConfigureServices` method.

Category

ASP.NET Core

Affected APIs

None

Target framework: .NET Framework support dropped

Starting with ASP.NET Core 3.0, .NET Framework is an unsupported target framework.

Change description

.NET Framework 4.8 is the last major version of .NET Framework. New ASP.NET Core apps should be built on .NET Core. Starting with the .NET Core 3.0 release, you can think of ASP.NET Core 3.0 as being part of .NET Core.

Customers using ASP.NET Core with .NET Framework can continue in a fully supported fashion using the 2.1 LTS release. Support and servicing for 2.1 continues until at least August 21, 2021. This date is three years after declaration of the LTS release per the .NET Support Policy. Support for ASP.NET Core 2.1 packages **on .NET Framework** will extend indefinitely, similar to the servicing policy for other package-based ASP.NET frameworks.

For more information about porting from .NET Framework to .NET Core, see Porting to .NET Core.

`Microsoft.Extensions` packages (such as logging, dependency injection, and configuration) and Entity Framework Core aren't affected. They'll continue to support .NET Standard.

For more information on the motivation for this change, see the original blog post.

Version introduced

3.0

Old behavior

ASP.NET Core apps could run on either .NET Core or .NET Framework.

New behavior

ASP.NET Core apps can only be run on .NET Core.

Recommended action

Take one of the following actions:

- Keep your app on ASP.NET Core 2.1.
- Migrate your app and dependencies to .NET Core.

Category

ASP.NET Core

Affected APIs

None

Core .NET libraries

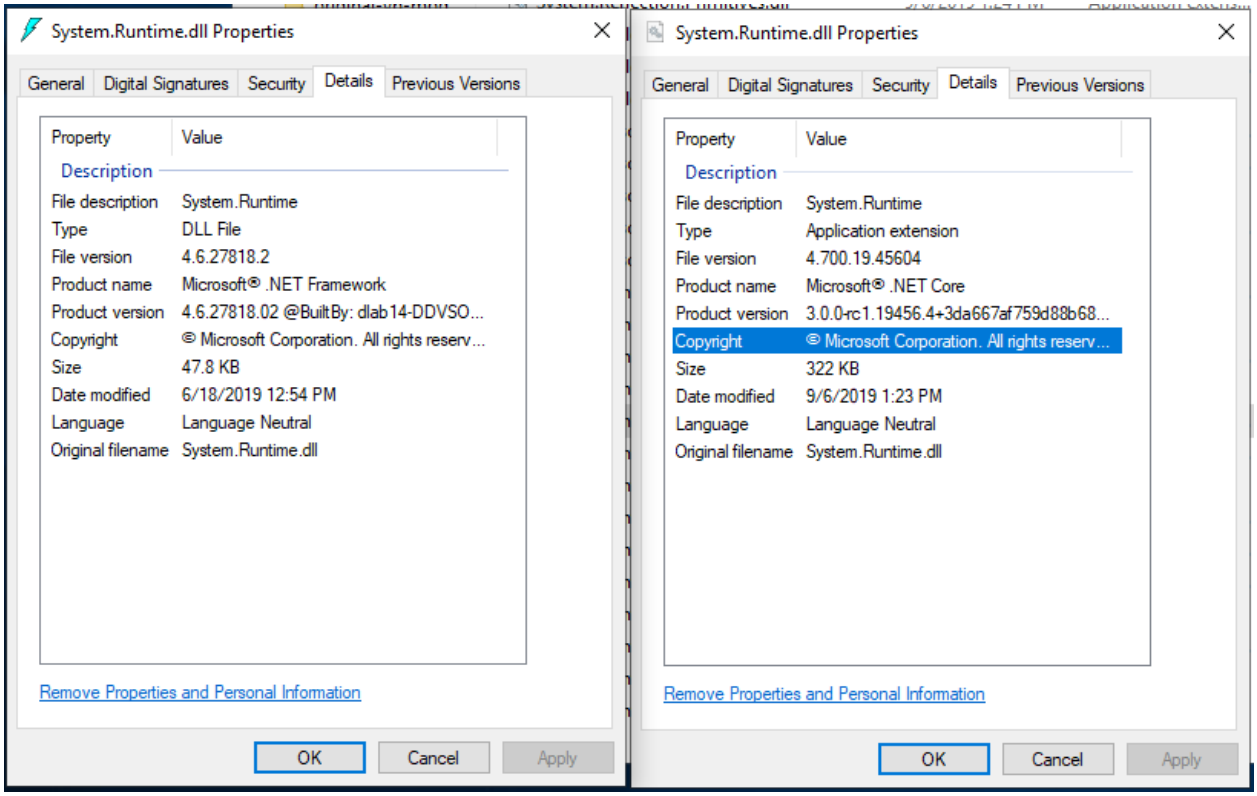
APIs that report version now report product and not file version

Many of the APIs that return versions in .NET Core now return the product version rather than the file version.

Change description

In .NET Core 2.2 and previous versions, methods such as Environment.Version, RuntimeInformation.FrameworkDescription, and the file properties dialog for .NET Core assemblies reflect the file version. Starting with .NET Core 3.0, they reflect the product version.

The following figure illustrates the difference in version information for the *System.Runtime.dll* assembly for .NET Core 2.2 (on the left) and .NET Core 3.0 (on the right) as displayed by the **Windows Explorer** file properties dialog.



Version introduced

3.0

Recommended action

None. This change should make version detection intuitive rather than obtuse.

Category

Core .NET libraries

Affected APIs

- Environment.Version
- RuntimeInformation.FrameworkDescription

Custom EncoderFallbackBuffer instances cannot fall back recursively

Custom EncoderFallbackBuffer instances cannot fall back recursively. The implementation of EncoderFallbackBuffer.GetNextChar() must result in a character sequence that is convertible to the destination encoding. Otherwise, an exception occurs.

Change description

During a character-to-byte transcoding operation, the runtime detects ill-formed or nonconvertible UTF-16 sequences and provides those characters to the EncoderFallbackBuffer.Fallback method. The `Fallback` method determines which characters should be substituted for the original nonconvertible data, and these characters are drained by calling EncoderFallbackBuffer.GetNextChar in a loop.

The runtime then attempts to transcode these substitution characters to the target encoding. If this operation succeeds, the runtime continues transcoding from where it left off in the original input string.

Previously, custom implementations of EncoderFallbackBuffer.GetNextChar() can return character sequences that are not convertible to the destination encoding. If the substituted characters cannot be transcoded to the target encoding, the runtime invokes the EncoderFallbackBuffer.Fallback method once again with the substitution characters, expecting the EncoderFallbackBuffer.GetNextChar() method to return a new substitution sequence. This process continues until the runtime eventually sees a well-formed, convertible substitution, or until a maximum recursion count is reached.

Starting with .NET Core 3.0, custom implementations of EncoderFallbackBuffer.GetNextChar() must return character sequences that are convertible to the destination encoding. If the substituted characters cannot be transcoded to the target encoding, an ArgumentException is thrown. The runtime will no longer make recursive calls into the EncoderFallbackBuffer instance.

This behavior only applies when all three of the following conditions are met:

- The runtime detects an ill-formed UTF-16 sequence or a UTF-16 sequence that cannot be converted to the target encoding.
- A custom EncoderFallback has been specified.
- The custom EncoderFallback attempts to substitute a new ill-formed or nonconvertible UTF-16 sequence.

Version introduced

3.0

Recommended action

Most developers needn't take any action.

If an application uses a custom EncoderFallback and EncoderFallbackBuffer class, ensure the implementation of EncoderFallbackBuffer.Fallback populates the fallback buffer with well-formed UTF-16 data that is directly convertible to the target encoding when the Fallback method is first invoked by the runtime.

Category

Core .NET libraries

Affected APIs

- EncoderFallbackBuffer.Fallback
- EncoderFallbackBuffer.GetNextChar()

Floating-point formatting and parsing behavior changed

Floating-point parsing and formatting behavior (by the Double and Single types) are now IEEE-compliant. This ensures that the behavior of floating-point types in .NET matches that of other IEEE-compliant languages. For example, `double.Parse("SomeLiteral")` should always match what C# produces for `double x = SomeLiteral`.

Change description

In .NET Core 2.2 and earlier versions, formatting with Double.ToString and Single.ToString, and parsing with Double.Parse, Double.TryParse, Single.Parse, and Single.TryParse are not IEEE-compliant. As a result, it's impossible to guarantee that a value will roundtrip with any supported standard or custom format string. For some inputs, the attempt to parse a formatted value can fail, and for others, the parsed value doesn't equal the original value.

Starting with .NET Core 3.0, floating-point parsing and formatting operations are IEEE 754-compliant.

The following table shows two code snippets and how the output changes between .NET Core 2.2 and .NET Core 3.1.

Code snippet	Output on .NET Core 2.2	Output on .NET Core 3.1
`Console.WriteLine((-0.0).ToString());`	0	-0

`var value = -3.123456789123456789;`
`Console.WriteLine(value ==`
`double.Parse(value.ToString()));`
	`False`	`True`

For more information, see the Floating-point parsing and formatting improvements in .NET Core 3.0 blog post.

Version introduced

3.0

Recommended action

The Potential impact to existing code section of the Floating-point parsing and formatting improvements in .NET Core 3.0 blog post suggests some changes you can make to your code if you want to maintain the previous behavior.

- For some differences in formatting, you can get behavior equivalent to the previous behavior by specifying a different format string.
- For differences in parsing, there's no mechanism to fall back to the previous behavior.

Category

Core .NET libraries

Affected APIs

- Double.ToString
- Single.ToString
- Double.Parse
- Double.TryParse
- Single.Parse
- Single.TryParse

Floating-point parsing operations no longer fail or throw an OverflowException

The floating-point parsing methods no longer throw an OverflowException or return `false` when they parse a string whose numeric value is outside the range of the Single or Double floating-point type.

Change description

In .NET Core 2.2 and earlier versions, the Double.Parse and Single.Parse methods throw an OverflowException for values that outside the range of their respective type. The Double.TryParse and Single.TryParse methods return `false` for the string representations of out-of-range numeric values.

Starting with .NET Core 3.0, the Double.Parse, Double.TryParse, Single.Parse, and Single.TryParse methods no longer fail when parsing out-of-range numeric strings. Instead, the Double parsing methods return Double.PositiveInfinity for values that exceed Double.MaxValue, and they return Double.NegativeInfinity for values that are less than Double.MinValue. Similarly, the Single parsing methods return Single.PositiveInfinity for values that exceed Single.MaxValue, and they return Single.NegativeInfinity for values that are less than Single.MinValue.

This change was made for improved IEEE 754:2008 compliance.

Version introduced

3.0

Recommended action

This change can affect your code in either of two ways:

- Your code depends on the handler for the OverflowException to execute when an overflow occurs. In this case, you should remove the `catch` statement and place any necessary code in an `If` statement that tests whether Double.IsInfinity or Single.IsInfinity is `true`.
- Your code assumes that floating-point values are not `Infinity`. In this case, you should add the necessary code to check for floating-point values of `PositiveInfinity` and `NegativeInfinity`.

Category

Core .NET libraries

Affected APIs

- Double.Parse
- Double.TryParse
- Single.Parse
- Single.TryParse

InvalidAsynchronousStateException moved to another assembly

The InvalidAsynchronousStateException class has been moved.

Change description

In .NET Core 2.2 and earlier versions, the InvalidAsynchronousStateException class is found in the *System.ComponentModel.TypeConverter* assembly.

Starting with .NET Core 3.0, it is found in the *System.ComponentModel.Primitives* assembly.

Version introduced

3.0

Recommended action

This change only affects applications that use reflection to load the InvalidAsynchronousStateException by calling a method such as Assembly.GetType or an overload of Activator.CreateInstance that assumes the type is in a particular assembly. If that is the case, update the assembly referenced in the method call to reflect the type's new assembly location.

Category

Core .NET libraries

Affected APIs

None.

Replacing ill-formed UTF-8 byte sequences follows Unicode guidelines

When the UTF8Encoding class encounters an ill-formed UTF-8 byte sequence during a byte-to-character transcoding operation, it replaces that sequence with a '�' (U+FFFD REPLACEMENT CHARACTER) character in the output string. .NET Core 3.0 differs from previous versions of .NET Core and the .NET Framework by following the Unicode best practice for performing this replacement during the transcoding operation.

This is part of a larger effort to improve UTF-8 handling throughout .NET, including by the new System.Text.Unicode.Utf8 and System.Text.Rune types. The UTF8Encoding type was given improved error handling mechanics so that it produces output consistent with the newly introduced types.

Change description

Starting with .NET Core 3.0, when transcoding bytes to characters, the UTF8Encoding class performs character substitution based on Unicode best practices. The substitution mechanism used is described by The Unicode Standard, Version 12.0, Sec. 3.9 (PDF) in the heading titled *U+FFFD Substitution of Maximal Subparts*.

This behavior *only* applies when the input byte sequence contains ill-formed UTF-8 data. Additionally, if the UTF8Encoding instance has been constructed with `throwOnInvalidBytes: true`, the `UTF8Encoding` instance will continue to throw on invalid input rather than perform U+FFFD replacement. For more information about the `UTF8Encoding` constructor, see UTF8Encoding(Boolean, Boolean).

The following table illustrates the impact of this change with an invalid 3-byte input:

Ill-formed 3-byte input	Output before .NET Core 3.0	Output starting with .NET Core 3.0
`[ ED A0 90 ]`	`[ FFFD FFFD ]` (2-character output)	`[ FFFD FFFD FFFD ]` (3-character output)

The 3-char output is the preferred output, according to *Table 3-9* of the previously linked Unicode Standard PDF.

Version introduced

3.0

Recommended action

No action is required on the part of the developer.

Category

Core .NET libraries

Affected APIs

- UTF8Encoding.GetCharCount
- UTF8Encoding.GetChars
- UTF8Encoding.GetString(Byte[], Int32, Int32)

TypeDescriptionProviderAttribute moved to another assembly

The TypeDescriptionProviderAttribute class has been moved.

Change description

In .NET Core 2.2 and earlier versions, The TypeDescriptionProviderAttribute class is found in the *System.ComponentModel.TypeConverter* assembly.

Starting with .NET Core 3.0, it is found in the *System.ObjectModel* assembly.

Version introduced

3.0

Recommended action

This change only affects applications that use reflection to load the TypeDescriptionProviderAttribute type by calling a method such as Assembly.GetType or an overload of Activator.CreateInstance that assumes the type is in a particular assembly. If that is the case, the assembly referenced in the method call should be updated to reflect the type's new assembly location.

Category

Windows Forms

Affected APIs

None.

ZipArchiveEntry no longer handles archives with inconsistent entry sizes

Zip archives list both compressed size and uncompressed size in the central directory and local header. The entry data itself also indicates its size. In .NET Core 2.2 and earlier versions, these values were never checked for consistency. Starting with .NET Core 3.0, they now are.

Change description

In .NET Core 2.2 and earlier versions, ZipArchiveEntry.Open() succeeds even if the local header disagrees with the central header of the zip file. Data is decompressed until the end of the compressed stream is reached, even if its length exceeds the uncompressed file size listed in the central directory/local header.

Starting with .NET Core 3.0, the ZipArchiveEntry.Open() method checks that local header and central header agree on compressed and uncompressed sizes of an entry. If they do not, the method throws an InvalidDataException if the archive's local header and/or data descriptor list sizes that disagree with the central directory of the zip file. When reading an entry, decompressed data is truncated to the uncompressed file size listed in the header.

This change was made to ensure that a ZipArchiveEntry correctly represents the size of its data and that only that amount of data is read.

Version introduced

3.0

Recommended action

Repackage any zip archive that exhibits these problems.

Category

Core .NET libraries

Affected APIs

- ZipArchiveEntry.Open()
- ZipFileExtensions.ExtractToDirectory
- ZipFileExtensions.ExtractToFile
- ZipFile.ExtractToDirectory

FieldInfo.SetValue throws exception for static, init-only fields

Starting in .NET Core 3.0, an exception is thrown when you attempt to set a value on a static, InitOnly field by calling System.Reflection.FieldInfo.SetValue.

Change description

In .NET Framework and versions of .NET Core prior to 3.0, you could set the value of a static field that's constant after it is initialized (readonly in C#) by calling System.Reflection.FieldInfo.SetValue. However, setting such a field in this way resulted in unpredictable behavior based on the target framework and optimization settings.

In .NET Core 3.0 and later versions, when you call SetValue on a static, InitOnly field, a System.FieldAccessException exception is thrown.

> **Tip**

An **InitOnly** field is one that can only be set at the time it's declared or in the constructor for the containing class. In other words, it's constant after it is initialized.

Version introduced

3.0

Recommended action

Initialize static, InitOnly fields in a static constructor. This applies to both dynamic and non-dynamic types.

Alternatively, you can remove the FieldAttributes.InitOnly attribute from the field, and then call FieldInfo.SetValue.

Category

Core .NET libraries

Affected APIs

- FieldInfo.SetValue(Object, Object)
- FieldInfo.SetValue(Object, Object, BindingFlags, Binder, CultureInfo)

Passing GroupCollection to extension methods taking IEnumerable<T> requires disambiguation

When calling an extension method that takes an `IEnumerable<T>` on a GroupCollection, you must disambiguate the type using a cast.

Change description

Starting in .NET Core 3.0, System.Text.RegularExpressions.GroupCollection implements `IEnumerable<KeyValuePair<String,Group>>` in addition to the other types it implements, including `IEnumerable<Group>`. This results in ambiguity when calling an extension method that takes an IEnumerable<T>. If you call such an extension method on a GroupCollection instance, for example, Enumerable.Count, you'll see the following compiler error:

CS1061: 'GroupCollection' does not contain a definition for 'Count' and no accessible extension method 'Count' accepting a first argument of type 'GroupCollection' could be found (are you missing a using directive or an assembly reference?)

In previous versions of .NET, there was no ambiguity and no compiler error.

Version introduced

3.0

Reason for change

This was an unintentional breaking change. Because it has been like this for some time, we don't plan to revert it. In addition, such a change would itself be breaking.

Recommended action

For GroupCollection instances, disambiguate calls to extension methods that accept an `IEnumerable<T>` with a cast.

```
C#

// Without a cast - causes CS1061.
match.Groups.Count(_ => true)

// With a disambiguating cast.
((IEnumerable<Group>)m.Groups).Count(_ => true);
```

Category

Core .NET libraries

Affected APIs

Any extension method that accepts an IEnumerable<T> is affected. For example:

- System.Collections.Immutable.ImmutableArray.ToImmutableArray<TSource>(IEnumerable<TSource>)
- System.Collections.Immutable.ImmutableDictionary.ToImmutableDictionary
- System.Collections.Immutable.ImmutableHashSet.ToImmutableHashSet
- System.Collections.Immutable.ImmutableList.ToImmutableList<TSource>(IEnumerable<TSource>)

- System.Collections.Immutable.ImmutableSortedDictionary.ToImmutableSortedDictionary
- System.Collections.Immutable.ImmutableSortedSet.ToImmutableSortedSet
- System.Data.DataTableExtensions.CopyToDataTable
- Most of the `System.Linq.Enumerable` methods, for example, System.Linq.Enumerable.Count
- System.Linq.ParallelEnumerable.AsParallel
- System.Linq.Queryable.AsQueryable

Cryptography

- BEGIN TRUSTED CERTIFICATE syntax no longer supported on Linux
- EnvelopedCms defaults to AES-256 encryption
- Minimum size for RSAOpenSsl key generation has increased
- .NET Core 3.0 prefers OpenSSL 1.1.x to OpenSSL 1.0.x
- CryptoStream.Dispose transforms final block only when writing

"BEGIN TRUSTED CERTIFICATE" syntax no longer supported for root certificates on Linux

Root certificates on Linux and other Unix-like systems (but not macOS) can be presented in two forms: the standard `BEGIN CERTIFICATE` PEM header, and the OpenSSL-specific `BEGIN TRUSTED CERTIFICATE` PEM header. The latter syntax allows for additional configuration that has caused compatibility issues with .NET Core's System.Security.Cryptography.X509Certificates.X509Chain class. `BEGIN TRUSTED CERTIFICATE` root certificate contents are no longer loaded by the chain engine starting in .NET Core 3.0.

Change description

Previously, both the `BEGIN CERTIFICATE` and `BEGIN TRUSTED CERTIFICATE` syntaxes were used to populate the root trust list. If the `BEGIN TRUSTED CERTIFICATE` syntax was used and additional options were specified in the file, X509Chain may have reported that the chain trust was explicitly disallowed (X509ChainStatusFlags.ExplicitDistrust). However, if the certificate was also specified with the `BEGIN CERTIFICATE` syntax in a previously loaded file, the chain trust was allowed.

Starting in .NET Core 3.0, `BEGIN TRUSTED CERTIFICATE` contents are no longer read. If the certificate is not also specified via a standard `BEGIN CERTIFICATE` syntax, the X509Chain

reports that the root is not trusted (X509ChainStatusFlags.UntrustedRoot).

Version introduced

3.0

Recommended action

Most applications are unaffected by this change, but applications that cannot see both root certificate sources because of permissions problems may experience unexpected `UntrustedRoot` errors after upgrading.

Many Linux distributions (or distros) write root certificates into two locations: a one-certificate-per-file directory, and a one-file concatenation. On some distros, the one-certificate-per-file directory uses the `BEGIN TRUSTED CERTIFICATE` syntax while the file concatenation uses the standard `BEGIN CERTIFICATE` syntax. Ensure that any custom root certificates are added as `BEGIN CERTIFICATE` in at least one of these locations, and that both locations can be read by your application.

The typical directory is */etc/ssl/certs/* and the typical concatenated file is */etc/ssl/cert.pem*. Use the command `openssl version -d` to determine the platform-specific root, which may differ from */etc/ssl/*. For example, on Ubuntu 18.04, the directory is */usr/lib/ssl/certs/* and the file is */usr/lib/ssl/cert.pem*. However, */usr/lib/ssl/certs/* is a symlink to */etc/ssl/certs/* and */usr/lib/ssl/cert.pem* does not exist.

Bash

```
$ openssl version -d
OPENSSLDIR: "/usr/lib/ssl"
$ ls -al /usr/lib/ssl
total 12
drwxr-xr-x  3 root root 4096 Dec 12 17:10 .
drwxr-xr-x 73 root root 4096 Feb 20 15:18 ..
lrwxrwxrwx  1 root root   14 Mar 27  2018 certs -> /etc/ssl/certs
drwxr-xr-x  2 root root 4096 Dec 12 17:10 misc
lrwxrwxrwx  1 root root   20 Nov 12 16:58 openssl.cnf ->
/etc/ssl/openssl.cnf
lrwxrwxrwx  1 root root   16 Mar 27  2018 private -> /etc/ssl/private
```

Category

Cryptography

Affected APIs

- System.Security.Cryptography.X509Certificates.X509Chain

EnvelopedCms defaults to AES-256 encryption

The default symmetric encryption algorithm used by `EnvelopedCms` has changed from TripleDES to AES-256.

Change description

In previous versions, when EnvelopedCms is used to encrypt data without specifying a symmetric encryption algorithm via a constructor overload, the data is encrypted with the TripleDES/3DES/3DEA/DES3-EDE algorithm.

Starting with .NET Core 3.0 (via version 4.6.0 of the System.Security.Cryptography.Pkcs NuGet package), the default algorithm has been changed to AES-256 for algorithm modernization and to improve the security of default options. If a message recipient certificate has a (non-EC) Diffie-Hellman public key, the encryption operation may fail with a CryptographicException due to limitations in the underlying platform.

In the following sample code, the data is encrypted with TripleDES if running on .NET Core 2.2 or earlier. If running on .NET Core 3.0 or later, it's encrypted with AES-256.

C#

```
EnvelopedCms cms = new EnvelopedCms(content);
cms.Encrypt(recipient);
return cms.Encode();
```

Version introduced

3.0

Recommended action

If you are negatively impacted by the change, you can restore TripleDES encryption by explicitly specifying the encryption algorithm identifier in an EnvelopedCms constructor that includes a parameter of type AlgorithmIdentifier, such as:

C#

```
Oid tripleDesOid = new Oid("1.2.840.113549.3.7", null);
AlgorithmIdentifier tripleDesIdentifier = new
AlgorithmIdentifier(tripleDesOid);
EnvelopedCms cms = new EnvelopedCms(content, tripleDesIdentifier);

cms.Encrypt(recipient);
return cms.Encode();
```

Category

Cryptography

Affected APIs

- EnvelopedCms()
- EnvelopedCms(ContentInfo)
- EnvelopedCms(SubjectIdentifierType, ContentInfo)

Minimum size for RSAOpenSsl key generation has increased

The minimum size for generating new RSA keys on Linux has increased from 384-bit to 512-bit.

Change description

Starting with .NET Core 3.0, the minimum legal key size reported by the `LegalKeySizes` property on RSA instances from RSA.Create, RSAOpenSsl, and RSACryptoServiceProvider on Linux has increased from 384 to 512.

As a result, in .NET Core 2.2 and earlier versions, a method call such as `RSA.Create(384)` succeeds. In .NET Core 3.0 and later versions, the method call `RSA.Create(384)` throws an exception indicating the size is too small.

This change was made because OpenSSL, which performs the cryptographic operations on Linux, raised its minimum between versions 1.0.2 and 1.1.0. .NET Core 3.0 prefers OpenSSL 1.1.x to 1.0.x, and the minimum reported version was raised to reflect this new higher dependency limitation.

Version introduced

3.0

Recommended action

If you call any of the affected APIs, ensure that the size of any generated keys is not less than the provider minimum.

> **Note**
>
> 384-bit RSA is already considered insecure (as is 512-bit RSA). Modern recommendations, such as **NIST Special Publication 800-57 Part 1 Revision 4**, suggest 2048-bit as the minimum size for newly generated keys.

Category

Cryptography

Affected APIs

- AsymmetricAlgorithm.LegalKeySizes
- RSA.Create
- RSAOpenSsl
- RSACryptoServiceProvider

.NET Core 3.0 prefers OpenSSL 1.1.x to OpenSSL 1.0.x

.NET Core for Linux, which works across multiple Linux distributions, can support both OpenSSL 1.0.x and OpenSSL 1.1.x. .NET Core 2.1 and .NET Core 2.2 look for 1.0.x first, then fall back to 1.1.x; .NET Core 3.0 looks for 1.1.x first. This change was made to add support for new cryptographic standards.

This change may impact libraries or applications that do platform interop with the OpenSSL-specific interop types in .NET Core.

Change description

In .NET Core 2.2 and earlier versions, the runtime prefers loading OpenSSL 1.0.x over 1.1.x. This means that the IntPtr and SafeHandle types for interop with OpenSSL are used with libcrypto.so.1.0.0 / libcrypto.so.1.0 / libcrypto.so.10 by preference.

Starting with .NET Core 3.0, the runtime prefers loading OpenSSL 1.1.x over OpenSSL 1.0.x, so the IntPtr and SafeHandle types for interop with OpenSSL are used with libcrypto.so.1.1 / libcrypto.so.11 / libcrypto.so.1.1.0 / libcrypto.so.1.1.1 by preference. As a result, libraries and applications that interoperate with OpenSSL directly may have incompatible pointers with the .NET Core-exposed values when upgrading from .NET Core 2.1 or .NET Core 2.2.

Version introduced

3.0

Recommended action

Libraries and applications that do direct operations with OpenSSL need to be careful to ensure they are using the same version of OpenSSL as the .NET Core runtime.

All libraries or applications that use IntPtr or SafeHandle values from the .NET Core cryptographic types directly with OpenSSL should compare the version of the library they use with the new SafeEvpPKeyHandle.OpenSslVersion property to ensure the pointers are compatible.

Category

Cryptography

Affected APIs

- SafeEvpPKeyHandle
- RSAOpenSsl(IntPtr)
- RSAOpenSsl(SafeEvpPKeyHandle)
- RSAOpenSsl.DuplicateKeyHandle()
- DSAOpenSsl(IntPtr)
- DSAOpenSsl(SafeEvpPKeyHandle)
- DSAOpenSsl.DuplicateKeyHandle()
- ECDsaOpenSsl(IntPtr)
- ECDsaOpenSsl(SafeEvpPKeyHandle)
- ECDsaOpenSsl.DuplicateKeyHandle()
- ECDiffieHellmanOpenSsl(IntPtr)
- ECDiffieHellmanOpenSsl(SafeEvpPKeyHandle)
- ECDiffieHellmanOpenSsl.DuplicateKeyHandle()
- X509Certificate.Handle

CryptoStream.Dispose transforms final block only when writing

The CryptoStream.Dispose method, which is used to finish `CryptoStream` operations, no longer attempts to transform the final block when reading.

Change description

In previous .NET versions, if a user performed an incomplete read when using CryptoStream in Read mode, the Dispose method could throw an exception (for example, when using AES with padding). The exception was thrown because the final block was attempted to be transformed but the data was incomplete.

In .NET Core 3.0 and later versions, Dispose no longer tries to transform the final block when reading, which allows for incomplete reads.

Reason for change

This change enables incomplete reads from the crypto stream when a network operation is canceled, without the need to catch an exception.

Version introduced

3.0

Recommended action

Most apps should not be affected by this change.

If your application previously caught an exception in case of an incomplete read, you can delete that `catch` block. If your app used transforming of the final block in hashing scenarios, you might need to ensure that the entire stream is read before it's disposed.

Category

Cryptography

Affected APIs

- System.Security.Cryptography.CryptoStream.Dispose

Entity Framework Core

Entity Framework Core breaking changes

Globalization

- "C" locale maps to the invariant locale

"C" locale maps to the invariant locale

.NET Core 2.2 and earlier versions depend on the default ICU behavior, which maps the "C" locale to the en_US_POSIX locale. The en_US_POSIX locale has an undesirable collation behavior, because it doesn't support case-insensitive string comparisons. Because some Linux distributions set the "C" locale as the default locale, users were experiencing unexpected behavior.

Change description

Starting with .NET Core 3.0, the "C" locale mapping has changed to use the Invariant locale instead of en_US_POSIX. The "C" locale to Invariant mapping is also applied to Windows for consistency.

Mapping "C" to en_US_POSIX culture caused customer confusion, because en_US_POSIX doesn't support case insensitive sorting/searching string operations. Because the "C" locale is used as a default locale in some of the Linux distros, customers experienced this undesired behavior on these operating systems.

Version introduced

3.0

Recommended action

Nothing specific more than the awareness of this change. This change affects only applications that use the "C" locale mapping.

Category

Globalization

Affected APIs

All collation and culture APIs are affected by this change.

MSBuild

- Resource manifest file name change

Resource manifest file name change

Starting in .NET Core 3.0, in the default case, MSBuild generates a different manifest file name for resource files.

Version introduced

3.0

Change description

Prior to .NET Core 3.0, if no `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata was specified for an `EmbeddedResource` item in the project file, MSBuild generated a manifest file name in the pattern `<RootNamespace>.<ResourceFilePathFromProjectRoot>.resources`. If `RootNamespace` is not defined in the project file, it defaults to the project name. For example, the generated manifest name for a resource file named *Form1.resx* in the root project directory was *MyProject.Form1.resources*.

Starting in .NET Core 3.0, if a resource file is colocated with a source file of the same name (for example, *Form1.resx* and *Form1.cs*), MSBuild uses type information from the source file to generate the manifest file name in the pattern `<Namespace>.<ClassName>.resources`. The namespace and class name are extracted from the first type in the colocated source file. For example, the generated manifest name for a resource file named *Form1.resx* that's colocated with a source file named *Form1.cs* is *MyNamespace.Form1.resources*. The key thing to note is that the first part of the file name is different to prior versions of .NET Core (*MyNamespace* instead of *MyProject*).

> **Note**

> If you have `LogicalName`, `ManifestResourceName`, or `DependentUpon` metadata specified on an `EmbeddedResource` item in the project file, then this change does not affect that resource file.

This breaking change was introduced with the addition of the `EmbeddedResourceUseDependentUponConvention` property to .NET Core projects. By default, resource files aren't explicitly listed in a .NET Core project file, so they have no `DependentUpon` metadata to specify how to name the generated *.resources* file. When `EmbeddedResourceUseDependentUponConvention` is set to `true`, which is the default, MSBuild looks for a colocated source file and extracts a namespace and class name from that file. If you set `EmbeddedResourceUseDependentUponConvention` to `false`, MSBuild generates the manifest name according to the previous behavior, which combines `RootNamespace` and the relative file path.

Recommended action

In most cases, no action is required on the part of the developer, and your app should continue to work. However, if this change breaks your app, you can either:

- Change your code to expect the new manifest name.
- Opt out of the new naming convention by setting `EmbeddedResourceUseDependentUponConvention` to `false` in your project file.

  XML

  ```
  <PropertyGroup>

  <EmbeddedResourceUseDependentUponConvention>false</EmbeddedResourceUseD
  ependentUponConvention>
  </PropertyGroup>
  ```

Category

MSBuild

Affected APIs

N/A

Networking

- Default value of HttpRequestMessage.Version changed to 1.1

Default value of HttpRequestMessage.Version changed to 1.1

The default value of the System.Net.Http.HttpRequestMessage.Version property has changed from 2.0 to 1.1.

Version introduced

3.0

Change description

In .NET Core 1.0 through 2.0, the default value of the System.Net.Http.HttpRequestMessage.Version property is 1.1. Starting with .NET Core 2.1, it was changed to 2.1.

Starting with .NET Core 3.0, the default version number returned by the System.Net.Http.HttpRequestMessage.Version property is once again 1.1.

Recommended action

Update your code if it depends on the System.Net.Http.HttpRequestMessage.Version property returning a default value of 2.0.

Category

Networking

Affected APIs

- System.Net.Http.HttpRequestMessage.Version

See also

- What's new in .NET Core 3.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Core 2.2

Article • 12/04/2021

.NET Core 2.2 includes enhancements in application deployment, event handling for runtime services, authentication to Azure SQL databases, JIT compiler performance, and code injection prior to the execution of the `Main` method.

New deployment mode

Starting with .NET Core 2.2, you can deploy framework-dependent executables, which are **.exe** files instead of **.dll** files. Functionally similar to framework-dependent deployments, framework-dependent executables (FDE) still rely on the presence of a shared system-wide version of .NET Core to run. Your app contains only your code and any third-party dependencies. Unlike framework-dependent deployments, FDEs are platform-specific.

This new deployment mode has the distinct advantage of building an executable instead of a library, which means you can run your app directly without invoking `dotnet` first.

Core

Handling events in runtime services

You may often want to monitor your application's use of runtime services, such as the GC, JIT, and ThreadPool, to understand how they impact your application. On Windows systems, this is commonly done by monitoring the ETW events of the current process. While this continues to work well, it's not always possible to use ETW if you're running in a low-privilege environment or on Linux or macOS.

Starting with .NET Core 2.2, CoreCLR events can now be consumed using the System.Diagnostics.Tracing.EventListener class. These events describe the behavior of such runtime services as GC, JIT, ThreadPool, and interop. These are the same events that are exposed as part of the CoreCLR ETW provider. This allows for applications to consume these events or use a transport mechanism to send them to a telemetry aggregation service. You can see how to subscribe to events in the following code sample:

C#

```
internal sealed class SimpleEventListener : EventListener
{
```

```
    // Called whenever an EventSource is created.
    protected override void OnEventSourceCreated(EventSource eventSource)
    {
        // Watch for the .NET runtime EventSource and enable all of its
events.
        if (eventSource.Name.Equals("Microsoft-Windows-DotNETRuntime"))
        {
            EnableEvents(eventSource, EventLevel.Verbose, (EventKeywords)
(-1));
        }
    }

    // Called whenever an event is written.
    protected override void OnEventWritten(EventWrittenEventArgs eventData)
    {
        // Write the contents of the event to the console.
        Console.WriteLine($"ThreadID = {eventData.OSThreadId} ID =
{eventData.EventId} Name = {eventData.EventName}");
        for (int i = 0; i < eventData.Payload.Count; i++)
        {
            string payloadString = eventData.Payload[i]?.ToString() ??
string.Empty;
            Console.WriteLine($"\tName = \"{eventData.PayloadNames[i]}\"
Value = \"{payloadString}\"");
        }
        Console.WriteLine("\n");
    }
}
```

In addition, .NET Core 2.2 adds the following two properties to the EventWrittenEventArgs class to provide additional information about ETW events:

- EventWrittenEventArgs.OSThreadId
- EventWrittenEventArgs.TimeStamp

Data

AAD authentication to Azure SQL databases with the SqlConnection.AccessToken property

Starting with .NET Core 2.2, an access token issued by Azure Active Directory can be used to authenticate to an Azure SQL database. To support access tokens, the AccessToken property has been added to the SqlConnection class. To take advantage of AAD authentication, download version 4.6 of the System.Data.SqlClient NuGet package. In order to use the feature, you can obtain the access token value using the Active Directory Authentication Library for .NET contained in the Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package.

JIT compiler improvements

Tiered compilation remains an opt-in feature

In .NET Core 2.1, the JIT compiler implemented a new compiler technology, *tiered compilation*, as an opt-in feature. The goal of tiered compilation is improved performance. One of the important tasks performed by the JIT compiler is optimizing code execution. For little-used code paths, however, the compiler may spend more time optimizing code than the runtime spends executing unoptimized code. Tiered compilation introduces two stages in JIT compilation:

- A **first tier**, which generates code as quickly as possible.
- A **second tier**, which generates optimized code for those methods that are executed frequently. The second tier of compilation is performed in parallel for enhanced performance.

For information on the performance improvement that can result from tiered compilation, see Announcing .NET Core 2.2 Preview 2.

For information about opting in to tiered compilation, see Jit compiler improvements in What's new in .NET Core 2.1.

Runtime

Injecting code prior to executing the Main method

Starting with .NET Core 2.2, you can use a startup hook to inject code prior to running an application's Main method. Startup hooks make it possible for a host to customize the behavior of applications after they have been deployed without needing to recompile or change the application.

We expect hosting providers to define custom configuration and policy, including settings that potentially influence the load behavior of the main entry point, such as the System.Runtime.Loader.AssemblyLoadContext behavior. The hook can be used to set up tracing or telemetry injection, to set up callbacks for handling, or to define other environment-dependent behavior. The hook is separate from the entry point, so that user code doesn't need to be modified.

See Host startup hook for more information.

See also

- What's new in .NET Core 3.1
- What's new in ASP.NET Core 2.2
- New features in EF Core 2.2

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Core 2.1

Article • 09/15/2021

.NET Core 2.1 includes enhancements and new features in the following areas:

- Tooling
- Roll forward
- Deployment
- Windows Compatibility Pack
- JIT compilation improvements
- API changes

Tooling

The .NET Core 2.1 SDK (v 2.1.300), the tooling included with .NET Core 2.1, includes the following changes and enhancements:

Build performance improvements

A major focus of .NET Core 2.1 is improving build-time performance, particularly for incremental builds. These performance improvements apply to both command-line builds using `dotnet build` and to builds in Visual Studio. Some individual areas of improvement include:

- For package asset resolution, resolving only assets used by a build rather than all assets.
- Caching of assembly references.
- Use of long-running SDK build servers, which are processes that span across individual `dotnet build` invocations. They eliminate the need to JIT-compile large blocks of code every time `dotnet build` is run. Build server processes can be automatically terminated with the following command:

  .NET CLI

  ```
  dotnet buildserver shutdown
  ```

New CLI commands

A number of tools that were available only on a per project basis using `DotnetCliToolReference` are now available as part of the .NET Core SDK. These tools include:

- `dotnet watch` provides a file system watcher that waits for a file to change before executing a designated set of commands. For example, the following command automatically rebuilds the current project and generates verbose output whenever a file in it changes:

  .NET CLI

  ```
  dotnet watch -- --verbose build
  ```

  Note the `--` option that precedes the `--verbose` option. It delimits the options passed directly to the `dotnet watch` command from the arguments that are passed to the child `dotnet` process. Without it, the `--verbose` option applies to the `dotnet watch` command, not the `dotnet build` command.

  For more information, see Develop ASP.NET Core apps using dotnet watch.

- `dotnet dev-certs` generates and manages certificates used during development in ASP.NET Core applications.

- `dotnet user-secrets` manages the secrets in a user secret store in ASP.NET Core applications.

- `dotnet sql-cache` creates a table and indexes in a Microsoft SQL Server database to be used for distributed caching.

- `dotnet ef` is a tool for managing databases, DbContext objects, and migrations in Entity Framework Core applications. For more information, see EF Core .NET Command-line Tools.

Global Tools

.NET Core 2.1 supports *Global Tools* -- that is, custom tools that are available globally from the command line. The extensibility model in previous versions of .NET Core made custom tools available on a per project basis only by using `DotnetCliToolReference`.

To install a Global Tool, you use the dotnet tool install command. For example:

.NET CLI

```
dotnet tool install -g dotnetsay
```

Once installed, the tool can be run from the command line by specifying the tool name. For more information, see .NET Core Global Tools overview.

Tool management with the `dotnet tool` command

In .NET Core 2.1 SDK, all tools operations use the `dotnet tool` command. The following options are available:

- dotnet tool install to install a tool.
- dotnet tool update to uninstall and reinstall a tool, which effectively updates it.
- dotnet tool list to list currently installed tools.
- dotnet tool uninstall to uninstall currently installed tools.

Roll forward

All .NET Core applications starting with .NET Core 2.0 automatically roll forward to the latest *minor version* installed on a system.

Starting with .NET Core 2.0, if the version of .NET Core that an application was built with is not present at run time, the application automatically runs against the latest installed *minor version* of .NET Core. In other words, if an application is built with .NET Core 2.0, and .NET Core 2.0 is not present on the host system but .NET Core 2.1 is, the application runs with .NET Core 2.1.

> **ⓘ Important**
>
> This roll-forward behavior doesn't apply to preview releases. By default, it also doesn't apply to major releases, but this can be changed with the settings below.

You can modify this behavior by changing the setting for the roll-forward on no candidate shared framework. The available settings are:

- `0` - disable minor version roll-forward behavior. With this setting, an application built for .NET Core 2.0.0 will roll forward to .NET Core 2.0.1, but not to .NET Core 2.2.0 or .NET Core 3.0.0.

- `1` - enable minor version roll-forward behavior. This is the default value for the setting. With this setting, an application built for .NET Core 2.0.0 will roll forward to either .NET Core 2.0.1 or .NET Core 2.2.0, depending on which one is installed, but it will not roll forward to .NET Core 3.0.0.
- `2` - enable minor and major version roll-forward behavior. If set, even different major versions are considered, so an application built for .NET Core 2.0.0 will roll forward to .NET Core 3.0.0.

You can modify this setting in any of three ways:

- Set the `DOTNET_ROLL_FORWARD_ON_NO_CANDIDATE_FX` environment variable to the desired value.
- Add the following line with the desired value to the *.runtimeconfig.json* file:

  JSON

  ```
  "rollForwardOnNoCandidateFx" : 0
  ```

- When using the .NET Core CLI, add the following option with the desired value to a .NET Core command such as `run`:

  .NET CLI

  ```
  dotnet run --rollForwardOnNoCandidateFx=0
  ```

Patch version roll forward is independent of this setting and is done after any potential minor or major version roll forward is applied.

Deployment

Self-contained application servicing

`dotnet publish` now publishes self-contained applications with a serviced runtime version. When you publish a self-contained application with the .NET Core 2.1 SDK (v 2.1.300), your application includes the latest serviced runtime version known by that SDK. When you upgrade to the latest SDK, you'll publish with the latest .NET Core runtime version. This applies for .NET Core 1.0 runtimes and later.

Self-contained publishing relies on runtime versions on NuGet.org. You do not need to have the serviced runtime on your machine.

Using the .NET Core 2.0 SDK, self-contained applications are published with the .NET Core 2.0.0 runtime unless a different version is specified via the `RuntimeFrameworkVersion` property. With this new behavior, you'll no longer need to set this property to select a higher runtime version for a self-contained application. The easiest approach going forward is to always publish with .NET Core 2.1 SDK (v 2.1.300).

For more information, see Self-contained deployment runtime roll forward.

Windows Compatibility Pack

When you port existing code from the .NET Framework to .NET Core, you can use the Windows Compatibility Pack. It provides access to 20,000 more APIs than are available in .NET Core. These APIs include types in the System.Drawing namespace, the EventLog class, WMI, Performance Counters, Windows Services, and the Windows registry types and members.

JIT compiler improvements

.NET Core incorporates a new JIT compiler technology called *tiered compilation* (also known as *adaptive optimization*) that can significantly improve performance. Tiered compilation is an opt-in setting.

One of the important tasks performed by the JIT compiler is optimizing code execution. For little-used code paths, however, the compiler may spend more time optimizing code than the runtime spends running unoptimized code. Tiered compilation introduces two stages in JIT compilation:

- A **first tier**, which generates code as quickly as possible.
- A **second tier**, which generates optimized code for those methods that are executed frequently. The second tier of compilation is performed in parallel for enhanced performance.

You can opt into tiered compilation in either of two ways.

- To use tiered compilation in all projects that use the .NET Core 2.1 SDK, set the following environment variable:

  Console

  ```
  COMPlus_TieredCompilation="1"
  ```

- To use tiered compilation on a per-project basis, add the `<TieredCompilation>` property to the `<PropertyGroup>` section of the MSBuild project file, as the following example shows:

  XML

  ```
  <PropertyGroup>
      <!-- other property definitions -->

      <TieredCompilation>true</TieredCompilation>
  </PropertyGroup>
  ```

API changes

`Span<T>` and `Memory<T>`

.NET Core 2.1 includes some new types that make working with arrays and other types of memory much more efficient. The new types include:

- System.Span<T> and System.ReadOnlySpan<T>.
- System.Memory<T> and System.ReadOnlyMemory<T>.

Without these types, when passing such items as a portion of an array or a section of a memory buffer, you have to make a copy of some portion of the data before passing it to a method. These types provide a virtual view of that data that eliminates the need for the additional memory allocation and copy operations.

The following example uses a Span<T> and Memory<T> instance to provide a virtual view of 10 elements of an array.

C#

```
using System;

class Program
{
    static void Main()
    {
        int[] numbers = new int[100];
        for (int i = 0; i < 100; i++)
        {
            numbers[i] = i * 2;
        }

        var part = new Span<int>(numbers, start: 10, length: 10);
```

```
        foreach (var value in part)
            Console.Write($"{value}  ");
    }
}
// The example displays the following output:
//     20  22  24  26  28  30  32  34  36  38
```

Brotli compression

.NET Core 2.1 adds support for Brotli compression and decompression. Brotli is a general-purpose lossless compression algorithm that is defined in RFC 7932 and is supported by most web browsers and major web servers. You can use the stream-based System.IO.Compression.BrotliStream class or the high-performance span-based System.IO.Compression.BrotliEncoder and System.IO.Compression.BrotliDecoder classes. The following example illustrates compression with the BrotliStream class:

C#

```
public static Stream DecompressWithBrotli(Stream toDecompress)
{
    MemoryStream decompressedStream = new MemoryStream();
    using (BrotliStream decompressionStream = new BrotliStream(toDecompress,
CompressionMode.Decompress))
    {
        decompressionStream.CopyTo(decompressedStream);
    }
    decompressedStream.Position = 0;
    return decompressedStream;
}
```

The BrotliStream behavior is the same as DeflateStream and GZipStream, which makes it easy to convert code that calls these APIs to BrotliStream.

New cryptography APIs and cryptography improvements

.NET Core 2.1 includes numerous enhancements to the cryptography APIs:

- System.Security.Cryptography.Pkcs.SignedCms is available in the System.Security.Cryptography.Pkcs package. The implementation is the same as the SignedCms class in the .NET Framework.
- New overloads of the X509Certificate.GetCertHash and X509Certificate.GetCertHashString methods accept a hash algorithm identifier to enable callers to get certificate thumbprint values using algorithms other than SHA-1.

- New Span<T>-based cryptography APIs are available for hashing, HMAC, cryptographic random number generation, asymmetric signature generation, asymmetric signature processing, and RSA encryption.
- The performance of System.Security.Cryptography.Rfc2898DeriveBytes has improved by about 15% by using a Span<T>-based implementation.
- The new System.Security.Cryptography.CryptographicOperations class includes two new methods:
  - FixedTimeEquals takes a fixed amount of time to return for any two inputs of the same length, which makes it suitable for use in cryptographic verification to avoid contributing to timing side-channel information.
  - ZeroMemory is a memory-clearing routine that cannot be optimized.
- The static RandomNumberGenerator.Fill method fills a Span<T> with random values.
- The System.Security.Cryptography.Pkcs.EnvelopedCms is now supported on Linux and macOS.
- Elliptic-Curve Diffie-Hellman (ECDH) is now available in the System.Security.Cryptography.ECDiffieHellman class family. The surface area is the same as in the .NET Framework.
- The instance returned by RSA.Create can encrypt or decrypt with OAEP using a SHA-2 digest, as well as generate or validate signatures using RSA-PSS.

Sockets improvements

.NET Core includes a new type, System.Net.Http.SocketsHttpHandler, and a rewritten System.Net.Http.HttpMessageHandler, that form the basis of higher-level networking APIs. System.Net.Http.SocketsHttpHandler, for example, is the basis of the HttpClient implementation. In previous versions of .NET Core, higher-level APIs were based on native networking implementations.

The sockets implementation introduced in .NET Core 2.1 has a number of advantages:

- A significant performance improvement when compared with the previous implementation.
- Elimination of platform dependencies, which simplifies deployment and servicing.
- Consistent behavior across all .NET Core platforms.

SocketsHttpHandler is the default implementation in .NET Core 2.1. However, you can configure your application to use the older HttpClientHandler class by calling the AppContext.SetSwitch method:

```
C#

AppContext.SetSwitch("System.Net.Http.UseSocketsHttpHandler", false);
```

You can also use an environment variable to opt out of using sockets implementations based on SocketsHttpHandler. To do this, set the `DOTNET_SYSTEM_NET_HTTP_USESOCKETSHTTPHANDLER` to either `false` or 0.

On Windows, you can also choose to use System.Net.Http.WinHttpHandler, which relies on a native implementation, or the SocketsHttpHandler class by passing an instance of the class to the HttpClient constructor.

On Linux and macOS, you can only configure HttpClient on a per-process basis. On Linux, you need to deploy libcurl if you want to use the old HttpClient implementation. (It is installed with .NET Core 2.0.)

Breaking changes

For information about breaking changes, see Breaking changes for migration from version 2.0 to 2.1.

See also

- What's new in .NET Core 3.1
- New features in EF Core 2.1
- What's new in ASP.NET Core 2.1

Breaking changes in .NET Core 2.1

Article • 03/18/2023

If you're migrating to version 2.1 of .NET Core, the breaking changes listed in this article may affect your app.

Core .NET libraries

Path APIs don't throw an exception for invalid characters

APIs that involve file paths no longer validate path characters or throw an ArgumentException if an invalid character is found.

Change description

In .NET Framework and .NET Core 1.0 - 2.0, the methods listed in the Affected APIs section throw an ArgumentException if the path argument contains an invalid path character. Starting in .NET Core 2.1, these methods no longer check for invalid path characters or throw an exception if an invalid character is found.

Reason for change

Aggressive validation of path characters blocks some cross-platform scenarios. This change was introduced so that .NET does not try to replicate or predict the outcome of operating system API calls. For more information, see the System.IO in .NET Core 2.1 sneak peek blog post.

Version introduced

.NET Core 2.1

Recommended action

If your code relied on these APIs to check for invalid characters, you can add a call to Path.GetInvalidPathChars.

Affected APIs

- System.IO.Directory.CreateDirectory
- System.IO.Directory.Delete
- System.IO.Directory.EnumerateDirectories
- System.IO.Directory.EnumerateFiles
- System.IO.Directory.EnumerateFileSystemEntries
- System.IO.Directory.GetCreationTime(String)
- System.IO.Directory.GetCreationTimeUtc(String)
- System.IO.Directory.GetDirectories
- System.IO.Directory.GetDirectoryRoot(String)
- System.IO.Directory.GetFiles
- System.IO.Directory.GetFileSystemEntries
- System.IO.Directory.GetLastAccessTime(String)
- System.IO.Directory.GetLastAccessTimeUtc(String)
- System.IO.Directory.GetLastWriteTime(String)
- System.IO.Directory.GetLastWriteTimeUtc(String)
- System.IO.Directory.GetParent(String)
- System.IO.Directory.Move(String, String)
- System.IO.Directory.SetCreationTime(String, DateTime)
- System.IO.Directory.SetCreationTimeUtc(String, DateTime)
- System.IO.Directory.SetCurrentDirectory(String)
- System.IO.Directory.SetLastAccessTime(String, DateTime)
- System.IO.Directory.SetLastAccessTimeUtc(String, DateTime)
- System.IO.Directory.SetLastWriteTime(String, DateTime)
- System.IO.Directory.SetLastWriteTimeUtc(String, DateTime)
- System.IO.DirectoryInfo ctor
- System.IO.Directory.GetDirectories
- System.IO.Directory.GetFiles
- System.IO.DirectoryInfo.GetFileSystemInfos
- System.IO.File.AppendAllText
- System.IO.File.AppendAllTextAsync
- System.IO.File.Copy
- System.IO.File.Create
- System.IO.File.CreateText
- System.IO.File.Decrypt
- System.IO.File.Delete
- System.IO.File.Encrypt
- System.IO.File.GetAttributes(String)
- System.IO.File.GetCreationTime(String)
- System.IO.File.GetCreationTimeUtc(String)

- System.IO.File.GetLastAccessTime(String)
- System.IO.File.GetLastAccessTimeUtc(String)
- System.IO.File.GetLastWriteTime(String)
- System.IO.File.GetLastWriteTimeUtc(String)
- System.IO.File.Move
- System.IO.File.Open
- System.IO.File.OpenRead(String)
- System.IO.File.OpenText(String)
- System.IO.File.OpenWrite(String)
- System.IO.File.ReadAllBytes(String)
- System.IO.File.ReadAllBytesAsync(String, CancellationToken)
- System.IO.File.ReadAllLines(String)
- System.IO.File.ReadAllLinesAsync(String, CancellationToken)
- System.IO.File.ReadAllText(String)
- System.IO.File.ReadAllTextAsync(String, CancellationToken)
- System.IO.File.SetAttributes(String, FileAttributes)
- System.IO.File.SetCreationTime(String, DateTime)
- System.IO.File.SetCreationTimeUtc(String, DateTime)
- System.IO.File.SetLastAccessTime(String, DateTime)
- System.IO.File.SetLastAccessTimeUtc(String, DateTime)
- System.IO.File.SetLastWriteTime(String, DateTime)
- System.IO.File.SetLastWriteTimeUtc(String, DateTime)
- System.IO.File.WriteAllBytes(String, Byte[])
- System.IO.File.WriteAllBytesAsync(String, Byte[], CancellationToken)
- System.IO.File.WriteAllLines
- System.IO.File.WriteAllLinesAsync
- System.IO.File.WriteAllText
- System.IO.FileInfo ctor
- System.IO.FileInfo.CopyTo
- System.IO.FileInfo.MoveTo
- System.IO.FileStream ctor
- System.IO.Path.GetFullPath(String)
- System.IO.Path.IsPathRooted(String)
- System.IO.Path.GetPathRoot(String)
- System.IO.Path.ChangeExtension(String, String)
- System.IO.Path.GetDirectoryName(String)
- System.IO.Path.GetExtension(String)
- System.IO.Path.HasExtension(String)
- System.IO.Path.Combine

See also

- System.IO in .NET Core 2.1 sneak peek

Private fields added to built-in struct types

Private fields were added to certain struct types in reference assemblies. As a result, in C#, those struct types must always be instantiated by using the new operator or default literal.

Change description

In .NET Core 2.0 and previous versions, some provided struct types, for example, ConsoleKeyInfo, could be instantiated without using the `new` operator or default literal in C#. This was because the reference assemblies used by the C# compiler didn't contain the private fields for the structs. All private fields for .NET struct types are added to the reference assemblies starting in .NET Core 2.1.

For example, the following C# code compiles in .NET Core 2.0, but not in .NET Core 2.1:

```
C#

ConsoleKeyInfo key;    // Struct type

if (key.ToString() == "y")
{
    Console.WriteLine("Yes!");
}
```

In .NET Core 2.1, the previous code results in the following compiler error: **CS0165 - Use of unassigned local variable 'key'**

Version introduced

2.1

Recommended action

Instantiate struct types by using the `new` operator or default literal.

For example:

C#

```
ConsoleKeyInfo key = new ConsoleKeyInfo();    // Struct type.

if (key.ToString() == "y")
    Console.WriteLine("Yes!");
```

C#

```
ConsoleKeyInfo key = default;    // Struct type.

if (key.ToString() == "y")
    Console.WriteLine("Yes!");
```

Category

Core .NET libraries

Affected APIs

- System.ArraySegment<T>.Enumerator
- System.ArraySegment<T>
- System.Boolean
- System.Buffers.MemoryHandle
- System.Buffers.StandardFormat
- System.Byte
- System.Char
- System.Collections.DictionaryEntry
- System.Collections.Generic.Dictionary<TKey,TValue>.Enumerator
- System.Collections.Generic.Dictionary<TKey,TValue>.KeyCollection.Enumerator
- System.Collections.Generic.Dictionary<TKey,TValue>.ValueCollection.Enumerator
- System.Collections.Generic.HashSet<T>.Enumerator
- System.Collections.Generic.KeyValuePair<TKey,TValue>
- System.Collections.Generic.LinkedList<T>.Enumerator
- System.Collections.Generic.List<T>.Enumerator
- System.Collections.Generic.Queue<T>.Enumerator
- System.Collections.Generic.SortedDictionary<TKey,TValue>.Enumerator
- System.Collections.Generic.SortedDictionary<TKey,TValue>.KeyCollection.Enumerator
- System.Collections.Generic.SortedDictionary<TKey,TValue>.ValueCollection.Enumerator
- System.Collections.Generic.SortedSet<T>.Enumerator

- System.Collections.Generic.Stack<T>.Enumerator
- System.Collections.Immutable.ImmutableArray<T>.Enumerator
- System.Collections.Immutable.ImmutableArray<T>
- System.Collections.Immutable.ImmutableDictionary<TKey,TValue>.Enumerator
- System.Collections.Immutable.ImmutableHashSet<T>.Enumerator
- System.Collections.Immutable.ImmutableList<T>.Enumerator
- System.Collections.Immutable.ImmutableQueue<T>.Enumerator
- System.Collections.Immutable.ImmutableSortedDictionary<TKey,TValue>.Enumerator
- System.Collections.Immutable.ImmutableSortedSet<T>.Enumerator
- System.Collections.Immutable.ImmutableStack<T>.Enumerator
- System.Collections.Specialized.BitVector32.Section
- System.Collections.Specialized.BitVector32
- LazyMemberInfo
- System.ComponentModel.Design.Serialization.MemberRelationship
- System.ConsoleKeyInfo
- System.Data.SqlTypes.SqlBinary
- System.Data.SqlTypes.SqlBoolean
- System.Data.SqlTypes.SqlByte
- System.Data.SqlTypes.SqlDateTime
- System.Data.SqlTypes.SqlDecimal
- System.Data.SqlTypes.SqlDouble
- System.Data.SqlTypes.SqlGuid
- System.Data.SqlTypes.SqlInt16
- System.Data.SqlTypes.SqlInt32
- System.Data.SqlTypes.SqlInt64
- System.Data.SqlTypes.SqlMoney
- System.Data.SqlTypes.SqlSingle
- System.Data.SqlTypes.SqlString
- System.DateTime
- System.DateTimeOffset
- System.Decimal
- System.Diagnostics.CounterSample
- System.Diagnostics.SymbolStore.SymbolToken
- System.Diagnostics.Tracing.EventSource.EventData
- System.Diagnostics.Tracing.EventSourceOptions
- System.Double
- System.Drawing.CharacterRange
- System.Drawing.Point
- System.Drawing.PointF

- System.Drawing.Rectangle
- System.Drawing.RectangleF
- System.Drawing.Size
- System.Drawing.SizeF
- System.Guid
- System.HashCode
- System.Int16
- System.Int32
- System.Int64
- System.IntPtr
- System.IO.Pipelines.FlushResult
- System.IO.Pipelines.ReadResult
- System.IO.WaitForChangedResult
- System.Memory<T>
- System.ModuleHandle
- System.Net.Security.SslApplicationProtocol
- System.Net.Sockets.IPPacketInformation
- System.Net.Sockets.SocketInformation
- System.Net.Sockets.UdpReceiveResult
- System.Net.WebSockets.ValueWebSocketReceiveResult
- System.Nullable<T>
- System.Numerics.BigInteger
- System.Numerics.Complex
- System.Numerics.Vector<T>
- System.ReadOnlyMemory<T>
- System.ReadOnlySpan<T>.Enumerator
- System.ReadOnlySpan<T>
- System.Reflection.CustomAttributeNamedArgument
- System.Reflection.CustomAttributeTypedArgument
- System.Reflection.Emit.Label
- System.Reflection.Emit.OpCode
- System.Reflection.Metadata.ArrayShape
- System.Reflection.Metadata.AssemblyDefinition
- System.Reflection.Metadata.AssemblyDefinitionHandle
- System.Reflection.Metadata.AssemblyFile
- System.Reflection.Metadata.AssemblyFileHandle
- System.Reflection.Metadata.AssemblyFileHandleCollection.Enumerator
- System.Reflection.Metadata.AssemblyFileHandleCollection
- System.Reflection.Metadata.AssemblyReference
- System.Reflection.Metadata.AssemblyReferenceHandle

- System.Reflection.Metadata.AssemblyReferenceHandleCollection.Enumerator
- System.Reflection.Metadata.AssemblyReferenceHandleCollection
- System.Reflection.Metadata.Blob
- System.Reflection.Metadata.BlobBuilder.Blobs
- System.Reflection.Metadata.BlobContentId
- System.Reflection.Metadata.BlobHandle
- System.Reflection.Metadata.BlobReader
- System.Reflection.Metadata.BlobWriter
- System.Reflection.Metadata.Constant
- System.Reflection.Metadata.ConstantHandle
- System.Reflection.Metadata.CustomAttribute
- System.Reflection.Metadata.CustomAttributeHandle
- System.Reflection.Metadata.CustomAttributeHandleCollection.Enumerator
- System.Reflection.Metadata.CustomAttributeHandleCollection
- System.Reflection.Metadata.CustomAttributeNamedArgument<TType>
- System.Reflection.Metadata.CustomAttributeTypedArgument<TType>
- System.Reflection.Metadata.CustomAttributeValue<TType>
- System.Reflection.Metadata.CustomDebugInformation
- System.Reflection.Metadata.CustomDebugInformationHandle
- System.Reflection.Metadata.CustomDebugInformationHandleCollection.Enumerator
- System.Reflection.Metadata.CustomDebugInformationHandleCollection
- System.Reflection.Metadata.DeclarativeSecurityAttribute
- System.Reflection.Metadata.DeclarativeSecurityAttributeHandle
- System.Reflection.Metadata.DeclarativeSecurityAttributeHandleCollection.Enumerator
- System.Reflection.Metadata.DeclarativeSecurityAttributeHandleCollection
- System.Reflection.Metadata.Document
- System.Reflection.Metadata.DocumentHandle
- System.Reflection.Metadata.DocumentHandleCollection.Enumerator
- System.Reflection.Metadata.DocumentHandleCollection
- System.Reflection.Metadata.DocumentNameBlobHandle
- System.Reflection.Metadata.Ecma335.ArrayShapeEncoder
- System.Reflection.Metadata.Ecma335.BlobEncoder
- System.Reflection.Metadata.Ecma335.CustomAttributeArrayTypeEncoder
- System.Reflection.Metadata.Ecma335.CustomAttributeElementTypeEncoder
- System.Reflection.Metadata.Ecma335.CustomAttributeNamedArgumentsEncoder
- System.Reflection.Metadata.Ecma335.CustomModifiersEncoder
- System.Reflection.Metadata.Ecma335.EditAndContinueLogEntry
- System.Reflection.Metadata.Ecma335.ExceptionRegionEncoder

- System.Reflection.Metadata.Ecma335.FixedArgumentsEncoder
- System.Reflection.Metadata.Ecma335.GenericTypeArgumentsEncoder
- System.Reflection.Metadata.Ecma335.InstructionEncoder
- System.Reflection.Metadata.Ecma335.LabelHandle
- System.Reflection.Metadata.Ecma335.LiteralEncoder
- System.Reflection.Metadata.Ecma335.LiteralsEncoder
- System.Reflection.Metadata.Ecma335.LocalVariablesEncoder
- System.Reflection.Metadata.Ecma335.LocalVariableTypeEncoder
- System.Reflection.Metadata.Ecma335.MethodBodyStreamEncoder.MethodBody
- System.Reflection.Metadata.Ecma335.MethodBodyStreamEncoder
- System.Reflection.Metadata.Ecma335.MethodSignatureEncoder
- System.Reflection.Metadata.Ecma335.NamedArgumentsEncoder
- System.Reflection.Metadata.Ecma335.NamedArgumentTypeEncoder
- System.Reflection.Metadata.Ecma335.NameEncoder
- System.Reflection.Metadata.Ecma335.ParametersEncoder
- System.Reflection.Metadata.Ecma335.ParameterTypeEncoder
- System.Reflection.Metadata.Ecma335.PermissionSetEncoder
- System.Reflection.Metadata.Ecma335.ReturnTypeEncoder
- System.Reflection.Metadata.Ecma335.ScalarEncoder
- System.Reflection.Metadata.Ecma335.SignatureDecoder<TType,TGenericContext>
- System.Reflection.Metadata.Ecma335.SignatureTypeEncoder
- System.Reflection.Metadata.Ecma335.VectorEncoder
- System.Reflection.Metadata.EntityHandle
- System.Reflection.Metadata.EventAccessors
- System.Reflection.Metadata.EventDefinition
- System.Reflection.Metadata.EventDefinitionHandle
- System.Reflection.Metadata.EventDefinitionHandleCollection.Enumerator
- System.Reflection.Metadata.EventDefinitionHandleCollection
- System.Reflection.Metadata.ExceptionRegion
- System.Reflection.Metadata.ExportedType
- System.Reflection.Metadata.ExportedTypeHandle
- System.Reflection.Metadata.ExportedTypeHandleCollection.Enumerator
- System.Reflection.Metadata.ExportedTypeHandleCollection
- System.Reflection.Metadata.FieldDefinition
- System.Reflection.Metadata.FieldDefinitionHandle
- System.Reflection.Metadata.FieldDefinitionHandleCollection.Enumerator
- System.Reflection.Metadata.FieldDefinitionHandleCollection
- System.Reflection.Metadata.GenericParameter
- System.Reflection.Metadata.GenericParameterConstraint
- System.Reflection.Metadata.GenericParameterConstraintHandle

- System.Reflection.Metadata.GenericParameterConstraintHandleCollection.Enumerator
- System.Reflection.Metadata.GenericParameterConstraintHandleCollection
- System.Reflection.Metadata.GenericParameterHandle
- System.Reflection.Metadata.GenericParameterHandleCollection.Enumerator
- System.Reflection.Metadata.GenericParameterHandleCollection
- System.Reflection.Metadata.GuidHandle
- System.Reflection.Metadata.Handle
- System.Reflection.Metadata.ImportDefinition
- System.Reflection.Metadata.ImportDefinitionCollection.Enumerator
- System.Reflection.Metadata.ImportDefinitionCollection
- System.Reflection.Metadata.ImportScope
- System.Reflection.Metadata.ImportScopeCollection.Enumerator
- System.Reflection.Metadata.ImportScopeCollection
- System.Reflection.Metadata.ImportScopeHandle
- System.Reflection.Metadata.InterfaceImplementation
- System.Reflection.Metadata.InterfaceImplementationHandle
- System.Reflection.Metadata.InterfaceImplementationHandleCollection.Enumerator
- System.Reflection.Metadata.InterfaceImplementationHandleCollection
- System.Reflection.Metadata.LocalConstant
- System.Reflection.Metadata.LocalConstantHandle
- System.Reflection.Metadata.LocalConstantHandleCollection.Enumerator
- System.Reflection.Metadata.LocalConstantHandleCollection
- System.Reflection.Metadata.LocalScope
- System.Reflection.Metadata.LocalScopeHandle
- System.Reflection.Metadata.LocalScopeHandleCollection.ChildrenEnumerator
- System.Reflection.Metadata.LocalScopeHandleCollection.Enumerator
- System.Reflection.Metadata.LocalScopeHandleCollection
- System.Reflection.Metadata.LocalVariable
- System.Reflection.Metadata.LocalVariableHandle
- System.Reflection.Metadata.LocalVariableHandleCollection.Enumerator
- System.Reflection.Metadata.LocalVariableHandleCollection
- System.Reflection.Metadata.ManifestResource
- System.Reflection.Metadata.ManifestResourceHandle
- System.Reflection.Metadata.ManifestResourceHandleCollection.Enumerator
- System.Reflection.Metadata.ManifestResourceHandleCollection
- System.Reflection.Metadata.MemberReference
- System.Reflection.Metadata.MemberReferenceHandle
- System.Reflection.Metadata.MemberReferenceHandleCollection.Enumerator
- System.Reflection.Metadata.MemberReferenceHandleCollection

- System.Reflection.Metadata.MetadataStringComparer
- System.Reflection.Metadata.MethodDebugInformation
- System.Reflection.Metadata.MethodDebugInformationHandle
- System.Reflection.Metadata.MethodDebugInformationHandleCollection.Enumerator
- System.Reflection.Metadata.MethodDebugInformationHandleCollection
- System.Reflection.Metadata.MethodDefinition
- System.Reflection.Metadata.MethodDefinitionHandle
- System.Reflection.Metadata.MethodDefinitionHandleCollection.Enumerator
- System.Reflection.Metadata.MethodDefinitionHandleCollection
- System.Reflection.Metadata.MethodImplementation
- System.Reflection.Metadata.MethodImplementationHandle
- System.Reflection.Metadata.MethodImplementationHandleCollection.Enumerator
- System.Reflection.Metadata.MethodImplementationHandleCollection
- System.Reflection.Metadata.MethodImport
- System.Reflection.Metadata.MethodSignature<TType>
- System.Reflection.Metadata.MethodSpecification
- System.Reflection.Metadata.MethodSpecificationHandle
- System.Reflection.Metadata.ModuleDefinition
- System.Reflection.Metadata.ModuleDefinitionHandle
- System.Reflection.Metadata.ModuleReference
- System.Reflection.Metadata.ModuleReferenceHandle
- System.Reflection.Metadata.NamespaceDefinition
- System.Reflection.Metadata.NamespaceDefinitionHandle
- System.Reflection.Metadata.Parameter
- System.Reflection.Metadata.ParameterHandle
- System.Reflection.Metadata.ParameterHandleCollection.Enumerator
- System.Reflection.Metadata.ParameterHandleCollection
- System.Reflection.Metadata.PropertyAccessors
- System.Reflection.Metadata.PropertyDefinition
- System.Reflection.Metadata.PropertyDefinitionHandle
- System.Reflection.Metadata.PropertyDefinitionHandleCollection.Enumerator
- System.Reflection.Metadata.PropertyDefinitionHandleCollection
- System.Reflection.Metadata.ReservedBlob<THandle>
- System.Reflection.Metadata.SequencePoint
- System.Reflection.Metadata.SequencePointCollection.Enumerator
- System.Reflection.Metadata.SequencePointCollection
- System.Reflection.Metadata.SignatureHeader
- System.Reflection.Metadata.StandaloneSignature
- System.Reflection.Metadata.StandaloneSignatureHandle

- System.Reflection.Metadata.StringHandle
- System.Reflection.Metadata.TypeDefinition
- System.Reflection.Metadata.TypeDefinitionHandle
- System.Reflection.Metadata.TypeDefinitionHandleCollection.Enumerator
- System.Reflection.Metadata.TypeDefinitionHandleCollection
- System.Reflection.Metadata.TypeLayout
- System.Reflection.Metadata.TypeReference
- System.Reflection.Metadata.TypeReferenceHandle
- System.Reflection.Metadata.TypeReferenceHandleCollection.Enumerator
- System.Reflection.Metadata.TypeReferenceHandleCollection
- System.Reflection.Metadata.TypeSpecification
- System.Reflection.Metadata.TypeSpecificationHandle
- System.Reflection.Metadata.UserStringHandle
- System.Reflection.ParameterModifier
- System.Reflection.PortableExecutable.CodeViewDebugDirectoryData
- System.Reflection.PortableExecutable.DebugDirectoryEntry
- System.Reflection.PortableExecutable.PEMemoryBlock
- System.Reflection.PortableExecutable.SectionHeader
- System.Runtime.CompilerServices.AsyncTaskMethodBuilder<TResult>
- System.Runtime.CompilerServices.AsyncTaskMethodBuilder
- System.Runtime.CompilerServices.AsyncValueTaskMethodBuilder<TResult>
- System.Runtime.CompilerServices.AsyncValueTaskMethodBuilder
- System.Runtime.CompilerServices.AsyncVoidMethodBuilder
- System.Runtime.CompilerServices.ConfiguredTaskAwaitable<TResult>.ConfiguredTaskAwaiter
- System.Runtime.CompilerServices.ConfiguredTaskAwaitable<TResult>
- System.Runtime.CompilerServices.ConfiguredTaskAwaitable.ConfiguredTaskAwaiter
- System.Runtime.CompilerServices.ConfiguredTaskAwaitable
- System.Runtime.CompilerServices.ConfiguredValueTaskAwaitable<TResult>
- System.Runtime.CompilerServices.ConfiguredValueTaskAwaitable<TResult>.ConfiguredValueTaskAwaiter
- System.Runtime.CompilerServices.TaskAwaiter<TResult>
- System.Runtime.CompilerServices.TaskAwaiter
- System.Runtime.CompilerServices.ValueTaskAwaiter<TResult>
- System.Runtime.CompilerServices.ValueTaskAwaiter<TResult>
- System.Runtime.InteropServices.ArrayWithOffset
- System.Runtime.InteropServices.GCHandle
- System.Runtime.InteropServices.HandleRef
- System.Runtime.InteropServices.OSPlatform
- System.Runtime.InteropServices.WindowsRuntime.EventRegistrationToken

- System.Runtime.Serialization.SerializationEntry
- System.Runtime.Serialization.StreamingContext
- System.RuntimeArgumentHandle
- System.RuntimeFieldHandle
- System.RuntimeMethodHandle
- System.RuntimeTypeHandle
- System.SByte
- System.Security.Cryptography.CngProperty
- System.Security.Cryptography.ECCurve
- System.Security.Cryptography.HashAlgorithmName
- System.Security.Cryptography.X509Certificates.X509ChainStatus
- System.Security.Cryptography.Xml.X509IssuerSerial
- System.ServiceProcess.SessionChangeDescription
- System.Single
- System.Span<T>.Enumerator
- System.Span<T>
- System.Threading.AsyncFlowControl
- System.Threading.AsyncLocalValueChangedArgs<T>
- System.Threading.CancellationToken
- System.Threading.CancellationTokenRegistration
- System.Threading.LockCookie
- System.Threading.SpinLock
- System.Threading.SpinWait
- System.Threading.Tasks.Dataflow.DataflowMessageHeader
- System.Threading.Tasks.ParallelLoopResult
- System.Threading.Tasks.ValueTask<TResult>
- System.TimeSpan
- System.TimeZoneInfo.TransitionTime
- System.Transactions.TransactionOptions
- System.TypedReference
- System.TypedReference
- System.UInt16
- System.UInt32
- System.UInt64
- System.UIntPtr
- System.Windows.Forms.ColorDialog.Color
- System.Windows.Media.Animation.KeyTime
- System.Windows.Media.Animation.RepeatBehavior
- System.Xml.Serialization.XmlDeserializationEvents
- Windows.Foundation.Point

- Windows.Foundation.Rect
- Windows.Foundation.Size
- Windows.UI.Color
- Windows.UI.Xaml.Controls.Primitives.GeneratorPosition
- Windows.UI.Xaml.CornerRadius
- Windows.UI.Xaml.Duration
- Windows.UI.Xaml.GridLength
- Windows.UI.Xaml.Media.Matrix
- Windows.UI.Xaml.Media.Media3D.Matrix3D
- Windows.UI.Xaml.Thickness

OpenSSL versions on macOS

The .NET Core 3.0 and later runtimes on macOS now prefer OpenSSL 1.1.x versions to OpenSSL 1.0.x versions for the AesCcm, AesGcm, DSAOpenSsl, ECDiffieHellmanOpenSsl, ECDsaOpenSsl, RSAOpenSsl, and SafeEvpPKeyHandle types.

The .NET Core 2.1 runtime now supports OpenSSL 1.1.x versions, but still prefers OpenSSL 1.0.x versions.

Change description

Previously, the .NET Core runtime used OpenSSL 1.0.x versions on macOS for types that interact with OpenSSL. The most recent OpenSSL 1.0.x version, OpenSSL 1.0.2, is now out of support. To keep types that use OpenSSL on supported versions of OpenSSL, the .NET Core 3.0 and later runtimes now use newer versions of OpenSSL on macOS.

With this change, the behavior for the .NET Core runtimes on macOS is as follows:

- The .NET Core 3.0 and later version runtimes use OpenSSL 1.1.x, if available, and fall back to OpenSSL 1.0.x only if there's no 1.1.x version available.

  For callers that use the OpenSSL interop types with custom P/Invokes, follow the guidance in the SafeEvpPKeyHandle.OpenSslVersion remarks. Your app may crash if you don't check the OpenSslVersion value.

- The .NET Core 2.1 runtime uses OpenSSL 1.0.x, if available, and falls back to OpenSSL 1.1.x if there's no 1.0.x version available.

  The 2.1 runtime prefers the earlier version of OpenSSL because the SafeEvpPKeyHandle.OpenSslVersion property does not exist in .NET Core 2.1, so the OpenSSL version cannot be reliably determined at run time.

Version introduced

- .NET Core 2.1.16
- .NET Core 3.0.3
- .NET Core 3.1.2

Recommended action

- Uninstall OpenSSL version 1.0.2 if it's no longer needed.
- Install OpenSSL 1.1.x if you use the AesCcm, AesGcm, DSAOpenSsl, ECDiffieHellmanOpenSsl, ECDsaOpenSsl, RSAOpenSsl, or SafeEvpPKeyHandle types.
- If you use the OpenSSL interop types with custom P/Invokes, follow the guidance in the SafeEvpPKeyHandle.OpenSslVersion remarks.

Category

Core .NET libraries

Affected APIs

- System.Security.Cryptography.AesCcm
- System.Security.Cryptography.AesGcm
- System.Security.Cryptography.DSAOpenSsl
- System.Security.Cryptography.ECDiffieHellmanOpenSsl
- System.Security.Cryptography.ECDsaOpenSsl
- System.Security.Cryptography.RSAOpenSsl
- System.Security.Cryptography.SafeEvpPKeyHandle

MSBuild

- Project tools now included in SDK

Project tools now included in SDK

The .NET Core 2.1 SDK now includes common CLI tooling, and you no longer need to reference these tools from the project.

Change description

In .NET Core 2.0, projects reference external .NET tools with the `<DotNetCliToolReference>` project setting. In .NET Core 2.1, some of these tools are included with the .NET Core SDK, and the setting is no longer needed. If you include references to these tools in your project, you'll receive an error similar to the following: **The tool 'Microsoft.EntityFrameworkCore.Tools.DotNet' is now included in the .NET Core SDK.**

Tools now included in .NET Core 2.1 SDK:

<DotNetCliToolReference> value	Tool
`Microsoft.DotNet.Watcher.Tools`	`dotnet-watch`
`Microsoft.Extensions.SecretManager.Tools`	dotnet-user-secrets
`Microsoft.Extensions.Caching.SqlConfig.Tools`	dotnet-sql-cache
`Microsoft.EntityFrameworkCore.Tools.DotNet`	dotnet-ef

Version introduced

.NET Core SDK 2.1.300

Recommended action

Remove the `<DotNetCliToolReference>` setting from your project.

Category

MSBuild

Affected APIs

N/A

See also

- What's new in .NET Core 2.1

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Core 2.0

Article • 02/18/2022

.NET Core 2.0 includes enhancements and new features in the following areas:

- Tooling
- Language support
- Platform improvements
- API changes
- Visual Studio integration
- Documentation improvements

Tooling

dotnet restore runs implicitly

In previous versions of .NET Core, you had to run the dotnet restore command to download dependencies immediately after you created a new project with the dotnet new command, as well as whenever you added a new dependency to your project.

You don't have to run dotnet restore because it's run implicitly by all commands that require a restore to occur, such as `dotnet new`, `dotnet build`, `dotnet run`, `dotnet test`, `dotnet publish`, and `dotnet pack`. To disable implicit restore, use the `--no-restore` option.

The `dotnet restore` command is still useful in certain scenarios where explicitly restoring makes sense, such as continuous integration builds in Azure DevOps Services or in build systems that need to explicitly control when the restore occurs.

For information about how to manage NuGet feeds, see the dotnet restore documentation.

You can also disable the automatic invocation of `dotnet restore` by passing the `--no-restore` switch to the `new`, `run`, `build`, `publish`, `pack`, and `test` commands.

Retargeting to .NET Core 2.0

If the .NET Core 2.0 SDK is installed, projects that target .NET Core 1.x can be retargeted to .NET Core 2.0.

To retarget to .NET Core 2.0, edit your project file by changing the value of the `<TargetFramework>` element (or the `<TargetFrameworks>` element if you have more than one target in your project file) from 1.x to 2.0:

XML

```
<PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
 </PropertyGroup>
```

You can also retarget .NET Standard libraries to .NET Standard 2.0 the same way:

XML

```
<PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
 </PropertyGroup>
```

For more information about migrating your project to .NET Core 2.0, see Migrating from ASP.NET Core 1.x to ASP.NET Core 2.0.

Language support

In addition to supporting C# and F#, .NET Core 2.0 also supports Visual Basic.

Visual Basic

With version 2.0, .NET Core now supports Visual Basic 2017. You can use Visual Basic to create the following project types:

- .NET Core console apps
- .NET Core class libraries
- .NET Standard class libraries
- .NET Core unit test projects
- .NET Core xUnit test projects

For example, to create a Visual Basic "Hello World" application, do the following steps from the command line:

1. Open a console window, create a directory for your project, and make it the current directory.

2. Enter the command `dotnet new console -lang vb`.

The command creates a project file with a `.vbproj` file extension, along with a Visual Basic source code file named *Program.vb*. This file contains the source code to write the string "Hello World!" to the console window.

3. Enter the command `dotnet run`. The .NET Core CLI automatically compiles and executes the application, which displays the message "Hello World!" in the console window.

Support for C# 7.1

.NET Core 2.0 supports C# 7.1, which adds a number of new features, including:

- The `Main` method, the application entry point, can be marked with the async keyword.
- Inferred tuple names.
- Default expressions.

Platform improvements

.NET Core 2.0 includes a number of features that make it easier to install .NET Core and to use it on supported operating systems.

.NET Core for Linux is a single implementation

.NET Core 2.0 offers a single Linux implementation that works on multiple Linux distributions. .NET Core 1.x required that you download a distribution-specific Linux implementation.

You can also develop apps that target Linux as a single operating system. .NET Core 1.x required that you target each Linux distribution separately.

Support for the Apple cryptographic libraries

.NET Core 1.x on macOS required the OpenSSL toolkit's cryptographic library. .NET Core 2.0 uses the Apple cryptographic libraries and doesn't require OpenSSL, so you no longer need to install it.

API changes and library support

Support for .NET Standard 2.0

.NET Standard defines a versioned set of APIs that must be available on .NET implementations that comply with that version of the standard. .NET Standard is targeted at library developers. It aims to guarantee the functionality that is available to a library that targets a version of .NET Standard on each .NET implementation. .NET Core 1.x supports .NET Standard version 1.6; .NET Core 2.0 supports the latest version, .NET Standard 2.0. For more information, see .NET Standard.

.NET Standard 2.0 includes over 20,000 more APIs than were available in .NET Standard 1.6. Much of this expanded surface area results from incorporating APIs that are common to the .NET Framework and Xamarin into .NET Standard.

.NET Standard 2.0 class libraries can also reference .NET Framework class libraries, provided that they call APIs that are present in .NET Standard 2.0. No recompilation of the .NET Framework libraries is required.

For a list of the APIs that have been added to .NET Standard since its last version, .NET Standard 1.6, see .NET Standard 2.0 vs. 1.6 .

Expanded surface area

The total number of APIs available on .NET Core 2.0 has more than doubled in comparison with .NET Core 1.1.

And with the Windows Compatibility Pack porting from .NET Framework has also become much simpler.

Support for .NET Framework libraries

.NET Core code can reference existing .NET Framework libraries, including existing NuGet packages. Note that the libraries must use APIs that are found in .NET Standard.

Visual Studio integration

Visual Studio 2017 version 15.3 and in some cases Visual Studio for Mac offer a number of significant enhancements for .NET Core developers.

Retargeting .NET Core apps and .NET Standard libraries

If the .NET Core 2.0 SDK is installed, you can retarget .NET Core 1.x projects to .NET Core 2.0 and .NET Standard 1.x libraries to .NET Standard 2.0.

To retarget your project in Visual Studio, you open the **Application** tab of the project's properties dialog and change the **Target framework** value to **.NET Core 2.0** or **.NET Standard 2.0**. You can also change it by right-clicking on the project and selecting the **Edit *.csproj file** option. For more information, see the Tooling section earlier in this topic.

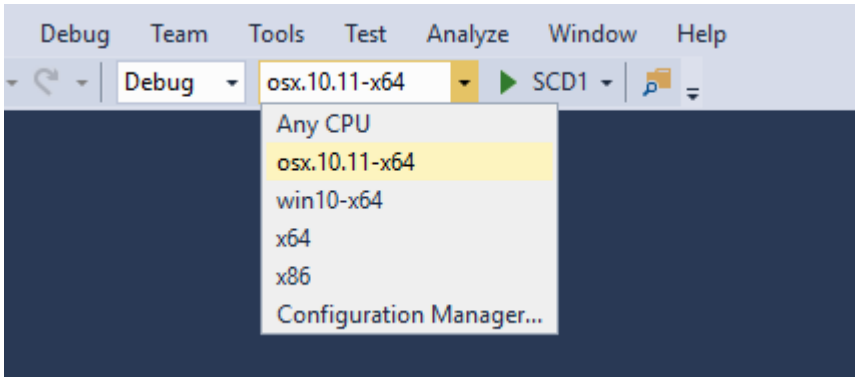
Live Unit Testing support for .NET Core

Whenever you modify your code, Live Unit Testing automatically runs any affected unit tests in the background and displays the results and code coverage live in the Visual Studio environment. .NET Core 2.0 now supports Live Unit Testing. Previously, Live Unit Testing was available only for .NET Framework applications.

For more information, see Live Unit Testing with Visual Studio and the Live Unit Testing FAQ.

Better support for multiple target frameworks

If you're building a project for multiple target frameworks, you can now select the target platform from the top-level menu. In the following figure, a project named SCD1 targets 64-bit macOS X 10.11 (`osx.10.11-x64`) and 64-bit Windows 10/Windows Server 2016 (`win10-x64`). You can select the target framework before selecting the project button, in this case to run a debug build.



Side-by-side support for .NET Core SDKs

You can now install the .NET Core SDK independently of Visual Studio. This makes it possible for a single version of Visual Studio to build projects that target different versions of .NET Core. Previously, Visual Studio and the .NET Core SDK were tightly coupled; a particular version of the SDK accompanied a particular version of Visual Studio.

Documentation improvements

.NET Application Architecture

.NET Application Architecture gives you access to a set of e-books that provide guidance, best practices, and sample applications when using .NET to build:

- Microservices and Docker containers
- Web applications with ASP.NET
- Mobile applications with Xamarin
- Applications that are deployed to the Cloud with Azure

See also

- What's new in ASP.NET Core 2.0

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET

.NET feedback

The .NET documentation is open source. Provide feedback here.

Open a documentation issue

Provide product feedback

What's new in .NET Standard

Article • 10/11/2022

.NET Standard is a formal specification that defines a versioned set of APIs that must be available on .NET implementations that comply with that version of the standard. .NET Standard is targeted at library developers. A library that targets a .NET Standard version can be used on any .NET or Xamarin implementation that supports that version of the standard.

.NET Standard is included with the .NET SDK. It's also included with Visual Studio if you select the .NET workload.

.NET Standard 2.1 is the last version of .NET Standard that will be released. For more information, see .NET 5+ and .NET Standard.

Supported .NET implementations

.NET Standard 2.1 is supported by the following .NET implementations:

- .NET Core 3.0 or later (including .NET 5 and later)
- Mono 6.4 or later
- Xamarin.iOS 12.16 or later
- Xamarin.Android 10.0 or later

.NET Standard 2.0 is supported by the following .NET implementations:

- .NET Core 2.0 or later (including .NET 5 and later)
- .NET Framework 4.6.1 or later
- Mono 5.4 or later
- Xamarin.iOS 10.14 or later
- Xamarin.Mac 3.8 or later
- Xamarin.Android 8.0 or later
- Universal Windows Platform 10.0.16299 or later

What's new in .NET Standard 2.1

.NET Standard 2.1 adds many APIs to the standard. Some of them are new APIs, and others are existing APIs that help to converge the .NET implementations even further. For a list of the APIs that have been added to .NET Standard 2.1, see .NET Standard 2.1 vs 2.0.

For more information, see the Announcing .NET Standard 2.1 blog post.

What's new in .NET Standard 2.0

.NET Standard 2.0 includes the following new features.

A vastly expanded set of APIs

Through version 1.6, .NET Standard included a comparatively small subset of APIs. Among those excluded were many APIs that were commonly used in .NET Framework or Xamarin. This complicates development, since it requires that developers find suitable replacements for familiar APIs when they develop applications and libraries that target multiple .NET implementations. .NET Standard 2.0 addresses this limitation by adding over 20,000 more APIs than were available in .NET Standard 1.6, the previous version of the standard. For a list of the APIs that have been added to .NET Standard 2.0, see .NET Standard 2.0 vs 1.6.

Some of the additions to the System namespace in .NET Standard 2.0 include:

- Support for the AppDomain class.
- Better support for working with arrays from additional members in the Array class.
- Better support for working with attributes from additional members in the Attribute class.
- Better calendar support and additional formatting options for DateTime values.
- Additional Decimal rounding functionality.
- Additional functionality in the Environment class.
- Enhanced control over the garbage collector through the GC class.
- Enhanced support for string comparison, enumeration, and normalization in the String class.
- Support for daylight saving adjustments and transition times in the TimeZoneInfo.AdjustmentRule and TimeZoneInfo.TransitionTime classes.
- Significantly enhanced functionality in the Type class.
- Better support for deserialization of exception objects by adding an exception constructor with SerializationInfo and StreamingContext parameters.

Support for .NET Framework libraries

Many libraries target .NET Framework rather than .NET Standard. However, most of the calls in those libraries are to APIs that are included in .NET Standard 2.0. Starting with .NET Standard 2.0, you can access .NET Framework libraries from a .NET Standard library

by using a compatibility shim. This compatibility layer is transparent to developers; you don't have to do anything to take advantage of .NET Framework libraries.

The single requirement is that the APIs called by the .NET Framework class library must be included in .NET Standard 2.0.

Support for Visual Basic

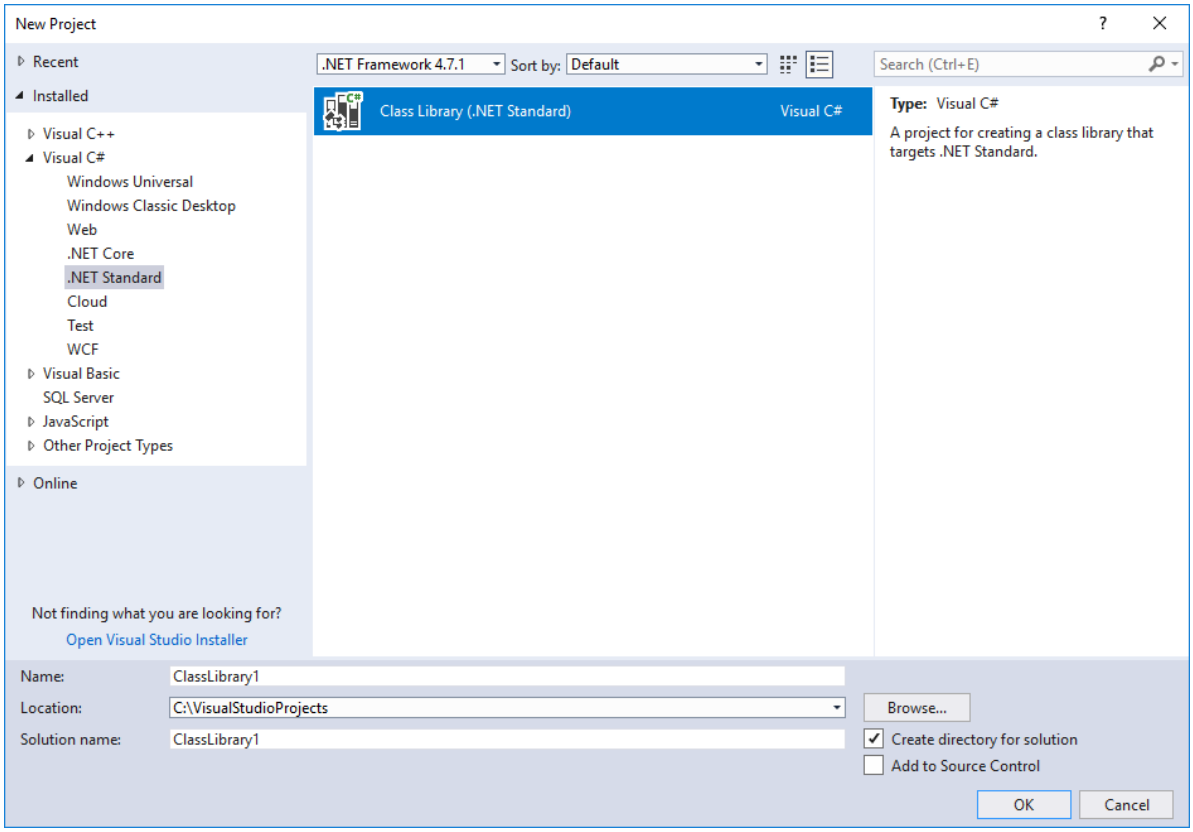
You can now develop .NET Standard libraries in Visual Basic. Visual Studio 2019 and Visual Studio 2017 version 15.3 or later with the .NET Core workload installed include a .NET Standard Class Library template. For Visual Basic developers who use other development tools and environments, you can use the dotnet new command to create a .NET Standard Library project. For more information, see the Tooling support for .NET Standard libraries.

Tooling support for .NET Standard libraries

With the release of .NET Core 2.0 and .NET Standard 2.0, both Visual Studio 2017 and the .NET CLI include tooling support for creating .NET Standard libraries.

If you install Visual Studio with the **.NET Core cross-platform development** workload, you can create a .NET Standard 2.0 library project by using a project template, as the following figure shows:

C#



If you're using the .NET CLI, the following dotnet new command creates a class library project that targets .NET Standard 2.0:

.NET CLI

```
dotnet new classlib
```

See also

- .NET Standard
- Introducing .NET Standard
- Download the .NET SDK

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

more information, see our contributor guide.