WIKIPEDIA

C Sharp (programming language)

C# (/si ʃɑːrp/ *see sharp*)[b] is a general-purpose, multi-paradigm programming language. C# encompasses static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.[15]

C# was designed by Anders Hejlsberg from Microsoft in 2000 and was later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO (ISO/IEC 23270) in 2003. Microsoft introduced C# along with .NET Framework and Visual Studio, both of which were closed-source. At the time, Microsoft had no open-source products. Four years later, in 2004, a free and open-source project called Mono began, providing a cross-platform compiler and runtime environment for the C# programming language. A decade later, Microsoft released Visual Studio Code (code editor), Roslyn (compiler), and the unified .NET platform (software framework), all of which support C# and are free, open-source, and cross-platform. Mono also joined Microsoft but was not merged into .NET.

As of 2021, the most recent version of the language is C# 10.0, which was released in 2021 in .NET 6.0.[16][17]

C#

Paradigm	Multi-paradigm: structured, imperative, object-oriented, event-driven, task-driven, functional, generic, reflective, concurrent
Family	C
Designed by	Anders Hejlsberg (Microsoft)
Developer	Mads Torgersen (Microsoft)
First appeared	2000[1]
Stable release	10.0[2] / 8 November 2021
Typing discipline	Static, dynamic,[3] strong, safe, nominative, partially inferred
Platform	Common Language Infrastructure
License	Roslyn compiler: MIT/X11[4] .NET Core CLR: MIT/X11[5] Mono compiler: dual GPLv3 and MIT/X11 DotGNU: dual GPL and LGPL

Contents

Filename extensions	`.cs`, `.csx`
Website	docs.microsoft.com/en-us/dotnet/csharp/ (https://docs.microsoft.com/en-us/dotnet/csharp/)
Major implementations	
Visual C#, .NET, .NET Framework (discontinued), Mono, DotGNU (discontinued), Universal Windows Platform	
Dialects	
Cω, Polyphonic C#, Enhanced C# (http://ecsharp.net)	
Influenced by	
C++,[6] Cω, Eiffel, F#,[a] Haskell, Icon, J#, J++, Java,[6] ML, Modula-3, Object Pascal,[7] VB	
Influenced	
Chapel,[8] Clojure,[9] Crystal,[10] D, J#, Dart,[11] F#, Hack, Java,[12][13] Kotlin, Nemerle, Oxygene, Rust, Swift,[14] Vala, TypeScript	
C Sharp Programming at Wikibooks	

Design goals

The Ecma standard lists these design goals for C#:[15]

- The language is intended to be a simple, modern, general-purpose, object-oriented programming language.
- The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.
- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Portability is very important for source code and programmers, especially those already familiar with C and C++.
- Support for internationalization is very important.
- C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
- Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.[18]

History

During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called *"Simple Managed C"* (SMC).[19][20] In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language".[21] Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.

Hejlsberg is C#'s principal designer and lead architect at Microsoft, and was previously involved with the design of Turbo Pascal, Embarcadero Delphi (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and Visual J++. In interviews and technical papers he has stated that flaws[22] in most major programming languages (e.g. C++, Java, Delphi, and Smalltalk) drove the fundamentals of the Common Language Runtime (CLR), which, in turn, drove the design of the C# language itself.

James Gosling, who created the Java programming language in 1994, and Bill Joy, a co-founder of Sun Microsystems, the originator of Java, called C# an "imitation" of Java; Gosling further said that "[C# is] sort of Java with reliability, productivity and security deleted."[23][24] Klaus Kreft and Angelika Langer (authors of a C++ streams book) stated in a blog post that "Java and C# are almost identical programming languages. Boring repetition that lacks innovation,"[25] "Hardly anybody will claim that Java or C# are revolutionary programming languages that changed the way we write programs," and "C# borrowed a lot from Java - and vice versa. Now that C# supports boxing and unboxing, we'll have a very similar feature in Java."[26] In July 2000, Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.[27]

Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming two quite different languages. One of the first major departures came with the addition of generics to both languages, with vastly different implementations. C# makes use of reification to provide "first-class" generic objects that can be used like any other class, with code generation performed at class-load time.[28] Furthermore, C# has added several major features to accommodate functional-style programming, culminating in the LINQ extensions released with C# 3.0 and its supporting framework of lambda expressions, extension methods, and anonymous types.[29] These features enable C# programmers to use functional programming techniques, such as closures, when it is advantageous to their application. The LINQ extensions and the functional imports help developers reduce the amount of boilerplate code that is included in common tasks like querying a database, parsing an xml file, or searching through a data structure, shifting the emphasis onto the actual program logic to help improve readability and maintainability.[30]

C# used to have a mascot called Andy (named after Anders Hejlsberg). It was retired on January 29, 2004.[31]

C# was originally submitted to the ISO subcommittee JTC 1/SC 22 for review,[32] under ISO/IEC 23270:2003,[33] was withdrawn and was then approved under ISO/IEC 23270:2006.[34] The 23270:2006 is withdrawn under 23270:2018 and approved with this version.[35]

Name

Microsoft first used the name C# in 1988 for a variant of the C language designed for incremental compilation.[36] That project was not completed but the name lives on.

The name "C sharp" was inspired by the musical notation whereby a sharp symbol indicates that the written note should be made a semitone higher in pitch.[37] This is similar to the language name of C++, where "++" indicates that a variable should be incremented by 1 after being evaluated. The sharp symbol also resembles a ligature of four "+" symbols (in a two-by-two grid), further implying that the language is an increment of C++.[38]

Due to technical limitations of display (standard fonts, browsers, etc.) and the fact that the sharp symbol (U+266F ♯ MUSIC SHARP SIGN (HTML &#9839; · &sharp;)) is not present on most keyboard layouts, the number sign (U+0023 # NUMBER SIGN (HTML &#35; · &num;)) was chosen to approximate the sharp symbol in the written name of the programming language.[39] This convention is reflected in the ECMA-334 C# Language Specification.[15]

C-sharp musical note

The "sharp" suffix has been used by a number of other .NET languages that are variants of existing languages, including J# (a .NET language also designed by Microsoft that is derived from Java 1.1), A# (from Ada), and the functional programming language F#.[40] The original implementation of Eiffel for .NET was called Eiffel#,[41] a name retired since the full Eiffel language is now supported. The suffix has also been used for libraries, such as Gtk# (a .NET wrapper for GTK and other GNOME libraries) and Cocoa# (a wrapper for Cocoa).

Versions

Version	Language specification			Date	.NET	Visual Studio
	Ecma	ISO/IEC	Microsoft			
C# 1.0	December 2002 (http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-334,%202nd%20edition,%20December%202002.pdf)	April 2003 (http://www.techstreet.com/cgi-bin/pdf/free/378672/ISO+IEC+23270-2003.pdf)	January 2002 (http://download.microsoft.com/download/a/9/e/a9e229b9-fee5-4c3e-8476-917dee385062/CSharp%20Language%20Specification%20v1.0.doc)	January 2002	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.1 C# 1.2			October 2003 (http://download.microsoft.com/download/5/e/5/5e58be0a-b02b-41ac-a4a3-7a22286214ff/csharp%20language%20specification%20v1.2.doc)	April 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0[42]	June 2006 (https://web.archive.org/web/20121202194727/http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf)	September 2006 (http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip)	September 2005 (http://download.microsoft.com/download/9/8/f/98fdf0c7-2bbd-40d3-9fd1-5a4159fa8044/csharp%202.0%20specification_sept_2005.doc)[c]	November 2005	.NET Framework 2.0 .NET Framework 3.0	Visual Studio 2005 Visual Studio 2008
C# 3.0[43]	None		August 2007 (http://download.microsoft.com/download/3/8/8/388e7205-bc10-4226-b2a8-75351c669b09/CSharp%20Language%20Specification.doc)	November 2007	.NET Framework 2.0 (Except LINQ)[44] .NET Framework 3.0 (Except LINQ)[44] .NET Framework 3.5	Visual Studio 2008
C# 4.0[45]			April 2010	April 2010	.NET Framework 4	Visual Studio 2010
C# 5.0[46]	December 2017 (https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf)	December 2018 (https://standards.iso.org/ittf/PubliclyAvailableStandards/c075178_ISO_IEC_23270_2018.zip)	June 2013 (https://www.microsoft.com/en-us/download/details.aspx?id=7029)	August 2012	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0[47]	None		Draft (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/lan	July 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1	Visual Studio 2015

		guage-specification/)			
C# 7.0[48][49]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-7.0/)	March 2017	.NET Framework 4.7	Visual Studio 2017 version 15.0
C# 7.1[50]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-7.1/)	August 2017	.NET Core 2.0	Visual Studio 2017 version 15.3[51]
C# 7.2[52]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-7.2/)	November 2017		Visual Studio 2017 version 15.5[53]
C# 7.3[54]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-7.3/)	May 2018	.NET Core 2.1 .NET Core 2.2 .NET Framework 4.8	Visual Studio 2017 version 15.7[53]
C# 8.0[55]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-8.0/)	September 2019	.NET Core 3.0 .NET Core 3.1	Visual Studio 2019 version 16.3[56]
C# 9.0[57]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-9.0/records)	September 2020	.NET 5.0	Visual Studio 2019 version 16.8[56]
C# 10.0[58]		Specification proposal (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-10.0/record-structs)	November 2021	.NET 6.0	Visual Studio 2022 version 17.0[59]

Syntax

The core syntax of the C# language is similar to that of other C-style languages such as C, C++ and Java, particularly:

- Semicolons are used to denote the end of a statement.
- Curly brackets are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into namespaces.
- Variables are assigned using an equals sign, but compared using two consecutive equals signs.
- Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

Distinguishing features

Some notable features of C# that distinguish it from C, C++, and Java where noted, are:

Portability

By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure (CLI).[60] Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or Fortran.

Typing

C# supports strongly, implicitly typed variable declarations with the keyword `var`, and implicitly typed arrays with the keyword `new[]` followed by a collection initializer.

C# supports a strict Boolean data type, `bool`. Statements that take conditions, such as `while` and `if`, require an expression of a type that implements the `true` operator, such as the Boolean type. While C++ also has a Boolean type, it can be freely converted to and from integers, and expressions such as `if (a)` require only that `a` is convertible to bool, allowing `a` to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly `bool` can prevent certain types of programming mistakes such as `if (a = b)` (use of assignment = instead of equality ==).

C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. No implicit conversions occur between Booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default.

C# has explicit support for covariance and contravariance in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.

Enumeration members are placed in their own scope.

The C# language does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.

Local variables cannot shadow variables of the enclosing block, unlike C and C++.

Metaprogramming

Metaprogramming can be achieved in several ways:

- Reflection using framework API
- Expression tree[61] language feature represents code as an abstract syntax tree, where each node is an expression that can be inspected or executed. This enables dynamic modification of executable code at runtime. Expression tree introduced some homoiconicity to the language.
- Attribute language feature are metadata attached to a field or a block of code like assemblies, members and types, and are equivalent to annotations in Java. Attributes are accessible to both the compiler and programmatically through reflection. Many of these attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.
- Source generators,[62] feature of Roslyn C# compiler, enable compile time metaprogramming. During the compilation process, developers can inspect the code being compiled (using compiler API) and generate new C# source files that can be added to the compilation.

Methods and functions

A method in C# is a member of a class that can be invoked as a function (a sequence of instructions), rather than the mere value-holding capability of a class property. As in other syntactically similar languages, such as C++ and ANSI C, the signature of a method is a declaration comprising in order: any optional accessibility keywords (such as `private`), the explicit specification of its return type (such as `int`, or the keyword `void` if no value is returned), the name of the method, and finally, a parenthesized sequence of comma-separated parameter specifications, each consisting of a parameter's type, its formal name and optionally, a default value to be used whenever none is provided. Certain specific kinds of methods, such as those that simply get or set a class property by return value or assignment, do not require a full signature, but in the general case, the definition of a class includes the full signature declaration of its methods.

Like C++, and unlike Java, C# programmers must use the scope modifier keyword `virtual` to allow methods to be overridden by subclasses.[63]

Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.

The type `dynamic` allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.

C# has support for strongly-typed function pointers via the keyword `delegate`. Like the Qt framework's pseudo-C++ *signal* and *slot*, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so.

C# offers Java-like `synchronized` method calls, via the attribute `[MethodImpl(MethodImplOptions.Synchronized)]`, and has support for mutually-exclusive locks via the keyword `lock`.

Property

C# supports classes with properties. The properties can be simple accessor functions with a backing field, or implement getter and setter functions.

Since C# 3.0 the syntactic sugar of auto-implemented properties is available,[64] where the accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class.

Namespace

A C# `namespace` provides the same level of code isolation as a Java `package` or a C++ `namespace`, with very similar rules and features to a `package`. Namespaces can be imported with the "using" syntax.[65]

Memory access

In C#, memory address pointers can only be used within blocks specifically marked as *unsafe*,[66] and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of an 'unmanaged' value type that does not contain any references to garbage-collected objects, array, string, or a block of stack-allocated memory. Code that is not marked as unsafe can still store and manipulate pointers through the `System.IntPtr` type, but it cannot dereference them.

Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed in most cases. Code that retains references to objects longer than is required can still experience higher memory usage than necessary, however once the final reference to an object is released the memory is available for garbage collection.

Exception

A range of standard exceptions are available to programmers. Methods in standard libraries regularly throw system exceptions in some circumstances and the range of exceptions thrown is normally documented. Custom exception classes can be defined for classes allowing specific handling to be put in place for particular circumstances as needed.[67]

Checked exceptions are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.[68]

Polymorphism

Unlike C++, C# does not support multiple inheritance, although a class can implement any number of "interfaces" (fully abstract classes). This was a design decision by the language's lead architect to avoid complications and to simplify architectural requirements throughout CLI.

When implementing multiple interfaces that contain a method with the same name and taking parameters of the same type in the same order (i.e. the same signature), similar to Java, C# allows both a single method to cover all interfaces and if necessary specific methods for each interface.

However, unlike Java, C# supports operator overloading.[69]

Language Integrated Query (LINQ)

C# has the ability to utilize LINQ through the .NET Framework. A developer can query a variety of data sources, provided `IEnumerable<T>` interface is implemented on the object. This includes XML documents, an ADO.NET dataset, and SQL databases.[70]

Using LINQ in C# brings advantages like Intellisense support, strong filtering capabilities, type safety with compile error checking ability, and consistency for querying data over a variety of sources.[71] There are several different language structures that can be utilized with C# and LINQ and they are query expressions, lambda expressions, anonymous types, implicitly typed variables, extension methods, and object initializers.[72]

Functional programming

Though primarily an imperative language, C# 2.0 offered limited support for functional programming through first-class functions and closures in the form of anonymous delegates.[73] C# 3.0 expanded support for functional programming with the introduction of a lightweight syntax for lambda expressions,[74] extension methods (an affordance for modules), and a list comprehension syntax in the form of a "query comprehension" language. C# 7.0 adds features typically found in functional languages like tuples, local functions and pattern matching.[75] C# 9.0 introduces record feature[76] which is primarily built for better supporting immutable data models.

Common type system

C# has a *unified type system*. This unified type system is called Common Type System (CTS).[77]

A unified type system implies that all types, including primitives such as integers, are subclasses of the `System.Object` class. For example, every type inherits a `ToString()` method.

Categories of data types

CTS separates data types into two categories:[77]

1. Reference types
2. Value types

Instances of value types neither have referential identity nor referential comparison semantics. Equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded. Value types are derived from `System.ValueType`, always have a default value, and can always be created and copied. Some other limitations on value types are that they cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor. Examples of value types are all primitive types, such as `int` (a signed 32-bit integer), `float` (a 32-bit IEEE floating-point number), `char` (a 16-bit Unicode code unit), and `System.DateTime` (identifies a specific point in time with nanosecond precision). Other examples are **`enum`** (enumerations) and **`struct`** (user defined structures).

In contrast, reference types have the notion of referential identity, meaning that each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same. This is reflected in default equality and inequality comparisons for reference types, which test for referential rather than structural equality, unless the corresponding operators are overloaded (such as the case for `System.String`). Some operations are not always possible, such as creating an instance of a reference type, copying an existing instance, or performing a value comparison on two existing instances. Though specific reference types can provide such services by exposing a public constructor or implementing a corresponding interface (such as `ICloneable` or `IComparable`). Examples of reference types are `object` (the ultimate base class for all other C# classes), `System.String` (a string of Unicode characters), and `System.Array` (a base class for all C# arrays).

Both type categories are extensible with user-defined types.

Boxing and unboxing

Boxing is the operation of converting a value-type object into a value of a corresponding reference type.[77] Boxing in C# is implicit.

Unboxing is the operation of converting a value of a reference type (previously boxed) into a value of a value type.[77] Unboxing in C# requires an explicit type cast. A boxed object of type T can only be unboxed to a T (or a nullable T).[78]

Example:

```
int foo = 42;         // Value type.
object bar = foo;     // foo is boxed to bar.
int foo2 = (int)bar;  // Unboxed back to value type.
```

Libraries

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the Common Language Infrastructure (CLI), which is standardized as ECMA-335 *Common Language Infrastructure (CLI)*.

In addition to the standard CLI specifications, there are many commercial and community class libraries that build on top of the .NET framework libraries to provide additional functionality.[79]

C# can make calls to any library included in the List of .NET libraries and frameworks.

Examples

Hello World

The following is a very simple C# program, a version of the classic "Hello world" example using the top-level statements feature introduced in C# 9:[80]

```
using System;

Console.WriteLine("Hello, world!");
```

For code written as C# 8 or lower, the entry point logic of a program must be written in a Main method inside a type:

```
using System;

// A version of the classic "Hello World" program
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

This code will display this text in the console window:

```
Hello, world!
```

Each line has a purpose:

```
using System;
```

The above line imports all types in the `System` namespace. For example, the `Console` class used later in the source code is defined in the `System` namespace, meaning it can be used without supplying the full name of the type (which includes the namespace).

```
// A version of the classic "Hello World" program
```

This line is a comment; it describes and documents the code for the programmer(s).

```
class Program
```

Above is a class definition for the `Program` class. Everything that follows between the pair of braces describes that class.

```
{
    ...
}
```

The curly brackets demarcate the boundaries of a code block. In this first instance, they are marking the start and end of the `Program` class.

```
static void Main(string[] args)
```

This declares the class member method where the program begins execution. The .NET runtime calls the `Main` method. Unlike in Java, the `Main` method does not need the `public` keyword, which tells the compiler that the method can be called from anywhere by any class.[81] Writing `static void Main(string[] args)` is equivalent to writing `private static void Main(string[] args)`. The static keyword makes the method accessible without an instance of `Program`. Each console application's `Main` entry point must be declared `static` otherwise the program would require an instance of `Program`, but any instance would require a program. To avoid that irresolvable circular dependency, C# compilers processing console applications (like that above) report an error if there is no `static Main` method. The `void` keyword declares that `Main` has no return value.

```
Console.WriteLine("Hello, world!");
```

This line writes the output. `Console` is a static class in the `System` namespace. It provides an interface to the standard input, output, and error streams for console applications. The program calls the `Console` method `WriteLine`, which displays on the console a line with the argument, the string `"Hello, world!"`.

GUI

A GUI example:

```
using System;
using System.Windows.Forms;

class Program
{
    static void Main()
    {
        MessageBox.Show("Hello, World!");
        Console.WriteLine("Is almost the same argument!");
    }
}
```

This example is similar to the previous example, except that it generates a dialog box that contains the message "Hello, World!" instead of writing it to the console.

Images

Another useful library is the `System.Drawing` library, which is used to programmatically draw images. For example:

```
using System;
using System.Drawing;

public class Example
{
    public static Image img;

    static void Main()
    {
        img = Image.FromFile("Image.png");
    }
}
```

This will create an image that is identical to that stored in "Image.png".

Standardization and licensing

In August 2001, Microsoft, Hewlett-Packard and Intel co-sponsored the submission of specifications for C# as well as the Common Language Infrastructure (CLI) to the standards organization Ecma International. In December 2001, ECMA released ECMA-334 *C# Language Specification*. C# became an ISO standard in 2003 (ISO/IEC 23270:2003 - *Information* *technology — Programming languages — C#*). ECMA had previously adopted equivalent specifications as the 2nd edition of C#, in December 2002. In June 2005, ECMA approved edition 3 of the C# specification, and updated ECMA-334. Additions included partial classes, anonymous methods, nullable types, and generics (somewhat similar to C++ templates). In July 2005, ECMA submitted to ISO/IEC JTC 1, via the latter's Fast-Track process, the standards and related TRs. This process usually takes 6–9 months.

The C# language definition and the CLI are standardized under ISO and Ecma standards that provide reasonable and non-discriminatory licensing protection from patent claims.

Microsoft initially agreed not to sue open-source developers for violating patents in non-profit projects for the part of the framework that is covered by the OSP.[82] Microsoft has also agreed not to enforce patents relating to Novell products against Novell's paying customers[83] with the exception of a list of products that do not explicitly mention C#, .NET or Novell's implementation of .NET (The Mono Project).[84] However, Novell maintained that Mono does not infringe any Microsoft patents.[85] Microsoft also made a specific agreement not to enforce patent rights related to the Moonlight browser plugin, which depends on Mono, provided it is obtained through Novell.[86]

A decade later, Microsoft began developing free, open-source, and cross-platform tooling for C#, namely Visual Studio Code, .NET Core, and Roslyn. Mono joined Microsoft as a project of Xamarin, a Microsoft subsidiary.

Implementations

Microsoft is leading the development of the open-source reference C# compilers and set of tools. The first compiler, Roslyn, compiles into intermediate language (IL), and the second one, RyuJIT,[87] is a JIT (just-in-time) compiler, which is dynamic and does on-the-fly optimization and compiles the IL into native code for the front-end of the CPU.[88] RyuJIT is open source and written in C++.[89] Roslyn is entirely written in managed code (C#), has been opened up and functionality surfaced as APIs. It is thus enabling developers to create refactoring and diagnostics tools.[4][90] Two branches of official implementation are .NET Framework (closed-source, Windows-only) and .NET Core (open-source, cross-platform); they eventually converged into one open-source implementation: .NET 5.0.[91] At .NET Framework 4.6, a new JIT compiler replaced the former.[87][92]

Other C# compilers (some of which include an implementation of the Common Language Infrastructure and .NET class libraries):

- Mono, a Microsoft-sponsored project provides an open-source C# compiler, a complete open-source implementation of the CLI (including the required framework libraries as they appear in the ECMA specification,) and a nearly complete implementation of the NET class libraries up to .NET Framework 3.5.
- The Elements tool chain from RemObjects includes RemObjects C#, which compiles C# code to .NET's Common Intermediate Language, Java bytecode, Cocoa, Android bytecode, WebAssembly, and native machine code for Windows, macOS, and Linux.
- The DotGNU project (now discontinued) also provided an open-source C# compiler, a nearly complete implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and subset of some of the remaining Microsoft proprietary .NET class libraries up to .NET 2.0 (those not documented or included in the ECMA specification, but included in Microsoft's standard .NET Framework distribution).

The Unity game engine uses C# as its primary scripting language. The Godot game engine has implemented an optional C# module thanks to a donation of $24,000 from Microsoft.[93]

See also

C# topics

- C# syntax
- Comparison of C# and Java
- Comparison of C# and Visual Basic .NET
- .NET standard libraries

IDEs

- Microsoft Visual Studio
- Microsoft Visual Studio Express
- Visual Studio Code
- MonoDevelop
- Morfik
- SharpDevelop
- Turbo C#
- Rider
- Xamarin Studio
- LINQPad

Notes

a. for async

b. By convention, a number sign is used for the second character in normal text; in artistic representations, sometimes a true sharp sign is used: C♯. However the ECMA 334 standard states: *"The name C# is written as the LATIN CAPITAL LETTER C (U+0043) followed by the NUMBER SIGN # (U+0023)."*

c. The Microsoft C# 2.0 specification document only contains the new 2.0 features. For older features, use the 1.2 specification above.

References

1. "InfoQ eMag: A Preview of C# 7" (https://www.infoq.com/minibooks/emag-c-sharp-preview).
2. https://devblogs.microsoft.com/dotnet/welcome-to-csharp-10/.
3. Torgersen, Mads (October 27, 2008). "New features in C# 4.0" (http://code.msdn.microsoft.com/csharpfuture/Release/ProjectReleases.aspx?ReleaseId=1686). Microsoft. Retrieved October 28, 2008.
4. "The Roslyn .NET compiler provides C# and Visual Basic languages with rich code analysis APIs.: dotnet/roslyn" (https://github.com/dotnet/roslyn). November 13, 2019 – via GitHub.
5. "CoreCLR is the runtime for .NET Core. It includes the garbage collector, JIT compiler, primitive data types and low-level classes.: dotnet/coreclr" (https://github.com/dotnet/coreclr). November 13, 2019 – via GitHub.
6. Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". *Journal of Computing Sciences in Colleges*. **22** (5). "Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java."
7. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#" (http://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/?pp=7). *Computerworld*. Retrieved February 12, 2010. "We all stand on the shoulders of giants here and every language builds on what went before it so we owe a lot to C, C++, Java, Delphi, all of these other things that came before us. (Anders Hejlsberg)"

8. "Chapel spec (Acknowledgments)" (http://chapel.cray.com/spec/spec-0.98.pdf) (PDF). Cray Inc. October 1, 2015. Retrieved January 14, 2016.
9. "Rich Hickey Q&A by Michael Fogus" (https://web.archive.org/web/20170111184835/http://www.codequarterly.com/2011/rich-hickey). Archived from the original (http://www.codequarterly.com/2011/rich-hickey) on January 11, 2017. Retrieved January 11, 2017.
10. Borenszweig, Ary (June 14, 2016). "Crystal 0.18.0 released!" (http://crystal-lang.org/2016/06/14/crystal-0.18.0-released.html#comment-2732771703). "It's heavily inspired by Ruby, and other languages (like C#, Go and Python)."
11. "Web Languages and VMs: Fast Code is Always in Fashion. (V8, Dart) - Google I/O 2013" (https://www.youtube.com/watch?v=huawCRlo9H4&t=30m10s). *YouTube*. Retrieved December 22, 2013.
12. Java 5.0 added several new language features (the enhanced for loop, autoboxing, varargs and annotations), after they were introduced in the similar (and competing) C# language [1] (http://www.barrycornelius.com/papers/java5/) [2] (http://www.levenez.com/lang/)
13. Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#" (http://www.barrycornelius.com/papers/java5/onefile/). University of Oxford Computing Services. Retrieved June 18, 2014. "In my opinion, it is C# that has caused these radical changes to the Java language. (Barry Cornelius)"
14. Lattner, Chris (June 3, 2014). "Chris Lattner's Homepage" (http://nondot.org/sabre/). Chris Lattner. Retrieved May 12, 2020. "The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list."
15. *C# Language Specification* (https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-334.pdf) (PDF) (4th ed.). Ecma International. June 2006. Retrieved January 26, 2012.
16. "Welcome to C# 10" (https://devblogs.microsoft.com/dotnet/welcome-to-csharp-10/). November 8, 2021.
17. "Announcing .NET 6 -- the Fastest .NET Yet" (https://devblogs.microsoft.com/dotnet/announcing-net-6/). November 8, 2021.
18. "Design Goals of C#" (https://www.java-samples.com/showtutorial.php?tutorialid=1425). *www.java-samples.com*. Retrieved October 6, 2021.
19. Zander, Jason (November 22, 2007). "Couple of Historical Facts" (https://docs.microsoft.com/en-us/archive/blogs/jasonz/couple-of-historical-facts). Retrieved February 23, 2009.
20. Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?" (https://web.archive.org/web/20160624010356/http://aspadvice.com/blogs/rbirkby/archive/2006/11/28/What-language-was-ASP.Net-originally-written-in_3F00_.aspx). Archived from the original (http://aspadvice.com/blogs/rbirkby/archive/2006/11/28/What-language-was-ASP.Net-originally-written-in_3F00_.aspx) on June 24, 2016. Retrieved February 21, 2008.
21. Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#" (http://www.computerworld.com.au/article/261958/-z_programming_languages_c). *Computerworld*. Retrieved October 1, 2008.
22. "Details" (http://nilsnaegele.com/techreview/Reviews/Details/1). *nilsnaegele.com*. Retrieved April 7, 2019.
23. Wylie Wong (2002). "Why Microsoft's C# isn't" (http://news.cnet.com/2100-1082-817522.html). CNET: CBS Interactive. Retrieved May 28, 2014.
24. Bill Joy (February 7, 2002). "Microsoft's blind spot" (https://www.cnet.com/news/microsofts-blind-spot/). cnet.com. Retrieved January 12, 2010.
25. Klaus Kreft and Angelika Langer (2003). "After Java and C# - what is next?" (http://www.artima.com/weblogs/viewpost.jsp?thread=6543). Retrieved June 18, 2013.

26. Klaus Kreft and Angelika Langer (July 3, 2003). "After Java and C# - what is next?" (http://www.artima.com/weblogs/viewpost.jsp?thread=6543). artima.com. Retrieved January 12, 2010.
27. Osborn, John (August 1, 2000). "Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg" (http://windowsdevcenter.com/pub/a/oreilly/windows/news/hejlsberg_0800.html). O'Reilly Media. Retrieved November 14, 2009.
28. "Generics (C# Programming Guide)" (http://msdn.microsoft.com/en-us/library/512aeb7t.aspx). Microsoft. Retrieved March 21, 2011.
29. Don Box and Anders Hejlsberg (February 2007). "LINQ: .NET Language-Integrated Query" (http://msdn.microsoft.com/en-us/library/bb308959.aspx). Microsoft. Retrieved March 21, 2011.
30. Mercer, Ian (April 15, 2010). "Why functional programming and LINQ is often better than procedural code" (http://blog.abodit.com/2010/04/why-functional-programming-is-better-linq-c-sharp-than-procedural-code/). abodit.com. Retrieved March 21, 2011.
31. "Andy Retires" (https://web.archive.org/web/20160119144858if_/http://blogs.msdn.com/b/danielfe/archive/2004/01/29/64429.aspx). *Dan Fernandez's Blog*. Blogs.msdn.com. January 29, 2004. Archived from the original (http://blogs.msdn.com/b/danielfe/archive/2004/01/29/64429.aspx) on January 19, 2016. Retrieved October 4, 2012.
32. "Technical committees - JTC 1/SC 22 - Programming languages, their environments and system software interfaces" (http://www.iso.org/iso/iso_technical_committee.html?commid=45202). ISO. Retrieved October 4, 2012.
33. "ISO/IEC 23270:2003 - Information technology - C# Language Specification" (https://web.archive.org/web/20120508100146/http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=36768). Iso.org. August 23, 2006. Archived from the original (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=36768) on May 8, 2012. Retrieved October 4, 2012.
34. "ISO/IEC 23270:2006 - Information technology - Programming languages - C#" (http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=42926). Iso.org. January 26, 2012. Retrieved October 4, 2012.
35. "SO/IEC 23270:2018 Information technology — Programming languages — C#" (https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/51/75178.html). *ISO*. Retrieved November 26, 2020.
36. Mariani, Rico. "My History of Visual Studio (Part 1) – Rico Mariani's Performance Tidbits" (https://blogs.msdn.microsoft.com/ricom/2009/10/05/my-history-of-visual-studio-part-1/). *Rico Mariani's Performance Tidbits*.
37. Kovacs, James (September 7, 2007). "C#/.NET History Lesson" (http://www.jameskovacs.com/blog/CNETHistoryLesson.aspx). Retrieved June 18, 2009.
38. Hejlsberg, Anders (October 1, 2008). "The A-Z of Programming Languages: C#" (http://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/?pp=2). *Computerworld*.
39. "Microsoft C# FAQ" (https://web.archive.org/web/20060214002638/http://msdn.microsoft.com/vcsharp/previous/2002/FAQ/default.aspx). Microsoft. Archived from the original (http://msdn.microsoft.com/vcsharp/previous/2002/FAQ/default.aspx) on February 14, 2006. Retrieved March 25, 2008.
40. "F# FAQ" (https://web.archive.org/web/20090218222543/http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/faq.aspx). Microsoft Research. Archived from the original (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/faq.aspx) on February 18, 2009. Retrieved June 18, 2009.
41. Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework" (http://msdn.microsoft.com/en-us/library/ms973898.aspx). Microsoft. Retrieved June 18, 2009.
42. "What's new in the C# 2.0 Language and Compiler" (https://web.archive.org/web/20101218191709/http://msdn.microsoft.com/en-us/library/7cz8t42e(v=vs.80).aspx). Microsoft. Archived from the original (http://msdn.microsoft.com/en-us/library/7cz8t42e(v=vs.80).aspx) on December 18, 2010. Retrieved June 11, 2014.

43. Hejlsberg, Anders; Torgersen, Mads. "Overview of C# 3.0" (http://msdn.microsoft.com/en-us/library/bb308966.aspx). *Microsoft Developer Network*. Microsoft. Retrieved June 11, 2014.
44. "Using C# 3.0 from .NET 2.0" (http://www.danielmoth.com/Blog/using-c-30-from-net-20.aspx). Danielmoth.com. May 13, 2007. Retrieved October 4, 2012.
45. Hejlsberg, Anders. "Future directions for C# and Visual Basic" (http://channel9.msdn.com/Events/BUILD/BUILD2011/TOOL-816T). *C# lead architect*. Microsoft. Retrieved September 21, 2011.
46. "An Introduction to New Features in C# 5.0" (http://blogs.msdn.com/b/mvpawardprogram/archive/2012/03/26/introduction-of-new-features-in-c-5-0.aspx). *MSDN Blogs*. Microsoft. Retrieved June 11, 2014.
47. "Language feature implementation status" (https://github.com/dotnet/roslyn/wiki/Languages-features-in-C%23-6-and-VB-14). *github*. Microsoft. Retrieved February 13, 2015.
48. "What's new in C# 7" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7). *Microsoft Docs*. December 21, 2016.
49. "New Features in C# 7.0" (https://blogs.msdn.microsoft.com/dotnet/2017/03/09/new-features-in-c-7-0/). *.NET Blog*. March 9, 2017. Retrieved June 9, 2017.
50. "What's new in C# 7.1" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-1). *Microsoft Docs*. Retrieved October 9, 2017.
51. "Visual Studio 2017 15.3 Release Notes" (https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes-v15.3). *docs.microsoft.com*.
52. "What's new in C# 7.2" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-2). *Microsoft Docs*. Retrieved November 26, 2017.
53. "Visual Studio 2017 15.9 Release Notes" (https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes). *docs.microsoft.com*.
54. "What's new in C# 7.3" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7-3). *Microsoft Docs*. Retrieved June 23, 2018.
55. "What's new in C# 8.0" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8). *Microsoft Docs*.
56. "Visual Studio 2019 16.8 Release Notes" (https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes). *docs.microsoft.com*.
57. BillWagner. "What's new in C# 9.0 - C# Guide" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9). *docs.microsoft.com*. Retrieved October 15, 2020.
58. "What's new in C# 10" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-10). *docs.microsoft.com*. Retrieved November 10, 2021.
59. "Visual Studio 2022 version 17.0 Release Notes" (https://docs.microsoft.com/en-us/visualstudio/releases/2022/release-notes). *docs.microsoft.com*.
60. *Visual Studio 2010 and .NET 4 Six-in-One*. Wrox Press. 2010. ISBN 978-0470499481.
61. BillWagner. "Expression Trees (C#)" (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/expression-trees/). *docs.microsoft.com*. Retrieved May 14, 2021.
62. "Introducing C# Source Generators" (https://devblogs.microsoft.com/dotnet/introducing-c-source-generators/). *.NET Blog*. April 29, 2020. Retrieved May 14, 2021.
63. "virtual (C# Reference)" (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual). *docs.microsoft.com*.
64. "Auto-Implemented Properties (C# Programming Guide)" (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties). Retrieved September 12, 2020.
65. "using directive - C# Reference" (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-directive). *Microsoft Docs*. Retrieved April 14, 2019.

66. BillWagner. "Unsafe code, pointers to data, and function pointers" (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/unsafe-code). *docs.microsoft.com*. Retrieved June 20, 2021.
67. "How to create user-defined exceptions" (https://docs.microsoft.com/en-us/dotnet/standard/exceptions/how-to-create-user-defined-exceptions). Retrieved September 12, 2020.
68. Venners, Bill; Eckel, Bruce (August 18, 2003). "The Trouble with Checked Exceptions" (http://www.artima.com/intv/handcuffs.html). Retrieved March 30, 2010.
69. BillWagner. "Operator overloading - C# reference" (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading). *docs.microsoft.com*. Retrieved June 20, 2021.
70. Zhang, Xue Dong; Teng, Zi Mu; Zhao, Dong Wang (September 2014). "Research of the Database Access Technology Under.NET Framework". *Applied Mechanics and Materials*. 644–650: 3077–3080. doi:10.4028/www.scientific.net/AMM.644-650.3077 (https://doi.org/10.4028%2Fwww.scientific.net%2FAMM.644-650.3077). S2CID 62201466 (https://api.semanticscholar.org/CorpusID:62201466). ProQuest 1565579768 (https://search.proquest.com/docview/1565579768).
71. Otey, Michael (February 2006). "LINQ to the Future". *SQL Server Magazine*. Vol. 8 no. 2. pp. 17–21. ProQuest 214859896 (https://search.proquest.com/docview/214859896).
72. Sheldon, William (November 2010). "New Features in LINQ". *SQL Server Magazine*. Vol. 12 no. 11. pp. 37–40. ProQuest 770609095 (https://search.proquest.com/docview/770609095).
73. BillWagner. "Anonymous functions - C# Programming Guide" (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/anonymous-functions). *docs.microsoft.com*. Retrieved May 15, 2021.
74. BillWagner. "Anonymous functions - C# Programming Guide" (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/anonymous-functions). *docs.microsoft.com*. Retrieved May 15, 2021.
75. "What's New in C# 7.0" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-7). *Microsoft Docs*. Retrieved April 14, 2019.
76. "C# 9.0 on the record" (https://devblogs.microsoft.com/dotnet/c-9-0-on-the-record/). *.NET Blog*. November 10, 2020. Retrieved May 15, 2021.
77. Archer, Tom (2001). "Part 2, Chapter 4: The Type System". *Inside C#*. Redmond, Washington: Microsoft Press. ISBN 0-7356-1288-9.
78. Lippert, Eric (March 19, 2009). "Representation and Identity" (http://blogs.msdn.com/b/ericlippert/archive/2009/03/19/representation-and-identity.aspx). *Fabulous Adventures In Coding*. Blogs.msdn.com. Retrieved October 4, 2012.
79. "Framework Libraries" (https://docs.microsoft.com/en-us/dotnet/standard/framework-libraries). *docs.microsoft.com*.
80. BillWagner. "What's new in C# 9.0 - C# Guide" (https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9). *docs.microsoft.com*. Retrieved May 14, 2021.
81. BillWagner. "Main() and command-line arguments" (https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/main-command-line). *docs.microsoft.com*. Retrieved August 5, 2021.
82. "Patent Pledge for Open Source Developers" (https://msdn.microsoft.com/en-us/openspecifications/dn646765).

83. "Patent Cooperation Agreement - Microsoft & Novell Interoperability Collaboration" (https://web.archive.org/web/20090517140252/http://www.microsoft.com/interop/msnovellcollab/patent_agreement.mspx). Microsoft. November 2, 2006. Archived from the original (http://www.microsoft.com/interop/msnovellcollab/patent_agreement.mspx) on May 17, 2009. Retrieved July 5, 2009. "Microsoft, on behalf of itself and its Subsidiaries (collectively "Microsoft"), hereby covenants not to sue Novell's Customers and Novell's Subsidiaries' Customers for infringement under Covered Patents of Microsoft on account of such a Customer's use of specific copies of a Covered Product as distributed by Novell or its Subsidiaries (collectively "Novell") for which Novell has received Revenue (directly or indirectly) for such specific copies; provided the foregoing covenant is limited to use by such Customer (i) of such specific copies that are authorized by Novell in consideration for such Revenue, and (ii) within the scope authorized by Novell in consideration for such Revenue."
84. "Definitions" (http://www.microsoft.com/interop/msnovellcollab/definitions2.aspx). Microsoft. November 2, 2006. Retrieved July 5, 2009.
85. Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community" (http://www.novell.com/linux/microsoft/faq_opensource.html). Retrieved July 5, 2009. "We maintain that Mono does not infringe any Microsoft patents."
86. "Covenant to Downstream Recipients of Moonlight - Microsoft & Novell Interoperability Collaboration" (https://web.archive.org/web/20100923213336/http://www.microsoft.com/interop/msnovellcollab/moonlight.mspx). Microsoft. September 28, 2007. Archived from the original (http://www.microsoft.com/interop/msnovellcollab/moonlight.mspx) on September 23, 2010. Retrieved March 8, 2008. ""Downstream Recipient" means an entity or individual that uses for its intended purpose a Moonlight Implementation obtained directly from Novell or through an Intermediate Recipient... Microsoft reserves the right to update (including discontinue) the foregoing covenant... "Moonlight Implementation" means only those specific portions of Moonlight 1.0 or Moonlight 1.1 that run only as a plug-in to a browser on a Personal Computer and are not licensed under GPLv3 or a Similar License."
87. "The RyuJIT transition is complete!" (https://devblogs.microsoft.com/dotnet/the-ryujit-transition-is-complete/). *microsoft.com*. June 19, 2018. Retrieved July 20, 2021.
88. "Managed Execution Process" (https://docs.microsoft.com/en-us/dotnet/standard/managed-execution-process). *microsoft.com*. Retrieved July 20, 2021.
89. "coreclr/src/jit/" (https://github.com/dotnet/coreclr/tree/master/src/jit). *github.com*. Retrieved July 20, 2021.
90. "C# Guide" (https://docs.microsoft.com/en-us/dotnet/csharp/). *docs.microsoft.com*.
91. "5.0.8" (https://dotnet.microsoft.com/download/dotnet/5.0). *microsoft.com*. Retrieved July 20, 2021.
92. "Mitigation: New 64-bit JIT Compiler" (https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/mitigation-new-64-bit-jit-compiler). *microsoft.com*. Retrieved July 20, 2021.
93. Etcheverry, Ignacio (October 21, 2017). "Introducing C# in Godot" (https://godotengine.org/article/introducing-csharp-godot). *Godot Engine*. Archived (https://web.archive.org/web/20181026084022/https://godotengine.org/article/introducing-csharp-godot) from the original on October 26, 2018. Retrieved October 26, 2018.

Further reading

- Drayton, Peter; Albahari, Ben; Neward, Ted (2002). *C# Language Pocket Reference* (https://archive.org/details/clanguagepocketr00pete). O'Reilly. ISBN 0-596-00429-X.
- Petzold, Charles (2002). *Programming Microsoft Windows with C#* (https://archive.org/details/isbn_9780735613706). Microsoft Press. ISBN 0-7356-1370-2.

External links

- C# Language Specification (https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/)
- C# Programming Guide (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index)
- ISO C# Language Specification (https://standards.iso.org/ittf/PubliclyAvailableStandards/c075178_ISO_IEC_23270_2018.zip)
- C# Compiler Platform ("Roslyn") source code (https://github.com/dotnet/roslyn)