

# The history of C#

Article • 03/07/2024

This article provides a history of each major release of the C# language. The C# team is continuing to innovate and add new features. Detailed language feature status, including features considered for upcoming releases can be found [on the dotnet/roslyn repository](#) on GitHub.

## Important

The C# language relies on types and methods in what the C# specification defines as a *standard library* for some of the features. The .NET platform delivers those types and methods in a number of packages. One example is exception processing. Every `throw` statement or expression is checked to ensure the object being thrown is derived from [Exception](#). Similarly, every `catch` is checked to ensure that the type being caught is derived from [Exception](#). Each version may add new requirements. To use the latest language features in older environments, you may need to install specific libraries. These dependencies are documented in the page for each specific version. You can learn more about the [relationships between language and library](#) for background on this dependency.

## C# version 12

*Released November 2023*

The following features were added in C# 12:

- [Primary constructors](#) - You can create primary constructors in any `class` or `struct` type.
- [Collection expressions](#) - A new syntax to specify collection expressions, including the spread element, (`..e`), to expand any collection.
- [Inline arrays](#) - Inline arrays enable you to create an array of fixed size in a `struct` type.
- [Optional parameters in lambda expressions](#) - You can define default values for parameters on lambda expressions.
- [ref readonly parameters](#) - `ref readonly` parameters enables more clarity for APIs that might be using `ref` parameters or `in` parameters.
- [Alias any type](#) - You can use the `using` alias directive to alias any type, not just named types.

- [Experimental attribute](#) - Indicate an experimental feature.

And, [Interceptors](#) - was released as a *Preview feature*.

Overall, C# 12 provides new features that make you more productive writing C# code. Syntax you already knew is available in more places. Other syntax enables consistency for related concepts.

## C# version 11

*Released November 2022*

The following features were added in C# 11:

- [Raw string literals](#)
- [Generic math support](#)
- [Generic attributes](#)
- [UTF-8 string literals](#)
- [Newlines in string interpolation expressions](#)
- [List patterns](#)
- [File-local types](#)
- [Required members](#)
- [Auto-default structs](#)
- [Pattern match `Span<char>` on a constant string](#)
- [Extended nameof scope](#)
- [Numeric IntPtr](#)
- [ref fields and scoped ref](#)
- [Improved method group conversion to delegate](#)
- [Warning wave 7](#)

C# 11 introduces *generic math* and several features that support that goal. You can write numeric algorithms once for all number types. There's more features to make working with `struct` types easier, like required members and auto-default structs. Working with strings gets easier with Raw string literals, newline in string interpolations, and UTF-8 string literals. Features like file local types enable source generators to be simpler. Finally, list patterns add more support for pattern matching.

## C# version 10

*Released November 2021*

C# 10 adds the following features and enhancements to the C# language:

- [Record structs](#)
- [Improvements of structure types](#)
- [Interpolated string handlers](#)
- [global using directives](#)
- [File-scoped namespace declaration](#)
- [Extended property patterns](#)
- [Improvements on lambda expressions](#)
- [Allow const interpolated strings](#)
- [Record types can seal ToString\(\)](#)
- [Improved definite assignment](#)
- [Allow both assignment and declaration in the same deconstruction](#)
- [Allow AsyncMethodBuilder attribute on methods](#)
- [CallerArgumentExpression attribute](#)
- [Enhanced #line pragma](#)

More features were available in *preview* mode. In order to use these features, you must set `<LangVersion>` to [Preview](#) in your project:

- [Generic attributes](#) later in this article.
- [static abstract members in interfaces](#).

C# 10 continues work on themes of removing ceremony, separating data from algorithms, and improved performance for the .NET Runtime.

Many of the features mean you type less code to express the same concepts. *Record structs* synthesize many of the same methods that *record classes* do. Structs and anonymous types support *with expressions*. *Global using directives* and *file scoped namespace declarations* mean you express dependencies and namespace organization more clearly. *Lambda improvements* make it easier to declare lambda expressions where they're used. New property patterns and deconstruction improvements create more concise code.

The new interpolated string handlers and `AsyncMethodBuilder` behavior can improve performance. These language features were applied in the .NET Runtime to achieve performance improvements in .NET 6.

C# 10 also marks more of a shift to the yearly cadence for .NET releases. Because not every feature can be completed in a yearly timeframe, you can try a couple of "preview" features in C# 10. Both *generic attributes* and *static abstract members in interfaces* can be used, but these preview features might change before their final release.

## C# version 9

C# 9 was released with .NET 5. It's the default language version for any assembly that targets the .NET 5 release. It contains the following new and enhanced features:

- [Records](#)
- [Init only setters](#)
- [Top-level statements](#)
- Pattern matching enhancements: [relational patterns](#) and [logical patterns](#)
- [Performance and interop](#)
  - [Native sized integers](#)
  - [Function pointers](#)
  - [Suppress emitting localsinit flag](#)
  - [Module initializers](#)
  - [New features for partial methods](#)
- [Fit and finish features](#)
  - [Target-typed new expressions](#)
  - [static anonymous functions](#)
  - [Target-typed conditional expressions](#)
  - [Covariant return types](#)
  - [Extension GetEnumerator support for foreach loops](#)
  - [Lambda discard parameters](#)
  - [Attributes on local functions](#)

C# 9 continues three of the themes from previous releases: removing ceremony, separating data from algorithms, and providing more patterns in more places.

[Top level statements](#) means your main program is simpler to read. There's less need for ceremony: a namespace, a `Program` class, and `static void Main()` are all unnecessary.

The introduction of [records](#) provides a concise syntax for reference types that follow value semantics for equality. You use these types to define data containers that typically define minimal behavior. [Init-only setters](#) provide the capability for nondestructive mutation (`with` expressions) in records. C# 9 also adds [covariant return types](#) so that derived records can override virtual methods and return a type derived from the base method's return type.

The [pattern matching](#) capabilities expanded in several ways. Numeric types now support *range patterns*. Patterns can be combined using `and`, `or`, and `not` patterns. Parentheses can be added to clarify more complex patterns:

C# 9 includes new pattern matching improvements:

- **Type patterns** match an object matches a particular type
- **Parenthesized patterns** enforce or emphasize the precedence of pattern combinations
- **Conjunctive and patterns** require both patterns to match
- **Disjunctive or patterns** require either pattern to match
- **Negated not patterns** require that a pattern doesn't match
- **Relational patterns** require the input be less than, greater than, less than or equal, or greater than or equal to a given constant

These patterns enrich the syntax for patterns. Consider these examples:

C#

```
public static bool IsLetter(this char c) =>
    c is >= 'a' and <= 'z' or >= 'A' and <= 'Z';
```

With optional parentheses to make it clear that `and` has higher precedence than `or`:

C#

```
public static bool IsLetterOrSeparator(this char c) =>
    c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z') or '.' or ',';
```

One of the most common uses is a new syntax for a null check:

C#

```
if (e is not null)
{
    // ...
}
```

Any of these patterns can be used in any context where patterns are allowed: `is` pattern expressions, `switch` expressions, nested patterns, and the pattern of a `switch` statement's `case` label.

Another set of features supports high-performance computing in C#:

- The `nint` and `nuint` types model the native-size integer types on the target CPU.
- **Function pointers** provide delegate-like functionality while avoiding the allocations necessary to create a delegate object.
- The `localsinit` instruction can be omitted to save instructions.

# Performance and interop

Another set of improvements supports scenarios where *code generators* add functionality:

- [Module initializers](#) are methods that the runtime calls when an assembly loads.
- [Partial methods](#) support new accessibility modifiers and non-void return types. In those cases, an implementation must be provided.

## Fit and finish features

C# 9 adds many other small features that improve developer productivity, both writing and reading code:

- Target-type `new` expressions
- `static` anonymous functions
- Target-type conditional expressions
- Extension `GetEnumerator()` support for `foreach` loops
- Lambda expressions can declare discard parameters
- Attributes can be applied to local functions

The C# 9 release continues the work to keep C# a modern, general-purpose programming language. Features continue to support modern workloads and application types.

## C# version 8.0

*Released September 2019*

C# 8.0 is the first major C# release that specifically targets .NET Core. Some features rely on new Common Language Runtime (CLR) capabilities, others on library types added only in .NET Core. C# 8.0 adds the following features and enhancements to the C# language:

- [Readonly members](#)
- [Default interface methods](#)
- [Pattern matching enhancements](#):
  - Switch expressions
  - Property patterns
  - Tuple patterns
  - Positional patterns
- [Using declarations](#)

- [Static local functions](#)
- [Disposable ref structs](#)
- [Nullable reference types](#)
- [Asynchronous streams](#)
- [Indices and ranges](#)
- [Null-coalescing assignment](#)
- [Unmanaged constructed types](#)
- [Stackalloc in nested expressions](#)
- [Enhancement of interpolated verbatim strings](#)

Default interface members require enhancements in the CLR. Those features were added in the CLR for .NET Core 3.0. Ranges and indexes, and asynchronous streams require new types in the .NET Core 3.0 libraries. Nullable reference types, while implemented in the compiler, is much more useful when libraries are annotated to provide semantic information regarding the null state of arguments and return values. Those annotations are being added in the .NET Core libraries.

## C# version 7.3

*Released May 2018*

There are two main themes to the C# 7.3 release. One theme provides features that enable safe code to be as performant as unsafe code. The second theme provides incremental improvements to existing features. New compiler options were also added in this release.

The following new features support the theme of better performance for safe code:

- You can access fixed fields without pinning.
- You can reassign `ref` local variables.
- You can use initializers on `stackalloc` arrays.
- You can use `fixed` statements with any type that supports a pattern.
- You can use more generic constraints.

The following enhancements were made to existing features:

- You can test `==` and `!=` with tuple types.
- You can use expression variables in more locations.
- You can attach attributes to the backing field of auto-implemented properties.
- Method resolution when arguments differ by `in` was improved.
- Overload resolution now has fewer ambiguous cases.

The new compiler options are:

- [-publicsign](#) to enable Open Source Software (OSS) signing of assemblies.
- [-pathmap](#) to provide a mapping for source directories.

## C# version 7.2

*Released November 2017*

C# 7.2 added several small language features:

- Initializers on `stackalloc` arrays.
- Use `fixed` statements with any type that supports a pattern.
- Access fixed fields without pinning.
- Reassign `ref` local variables.
- Declare `readonly struct` types, to indicate that a struct is immutable and should be passed as an `in` parameter to its member methods.
- Add the `in` modifier on parameters, to specify that an argument is passed by reference but not modified by the called method.
- Use the `ref readonly` modifier on method returns, to indicate that a method returns its value by reference but doesn't allow writes to that object.
- Declare `ref struct` types, to indicate that a struct type accesses managed memory directly and must always be stack allocated.
- Use additional generic constraints.
- [Non-trailing named arguments](#):
  - Positional arguments can follow named arguments.
- Leading underscores in numeric literals:
  - Numeric literals can now have leading underscores before any printed digits.
- [private protected access modifier](#):
  - The `private protected` access modifier enables access for derived classes in the same assembly.
- Conditional `ref` expressions:
  - The result of a conditional expression (`?:`) can now be a reference.

## C# version 7.1

*Released August 2017*

C# started releasing *point releases* with C# 7.1. This version added the [language version selection](#) configuration element, three new language features, and new compiler



behavior.

The new language features in this release are:

- [async Main method](#)
  - The entry point for an application can have the `async` modifier.
- [default literal expressions](#)
  - You can use default literal expressions in default value expressions when the target type can be inferred.
- Inferred tuple element names
  - The names of tuple elements can be inferred from tuple initialization in many cases.
- Pattern matching on generic type parameters
  - You can use pattern match expressions on variables whose type is a generic type parameter.

Finally, the compiler has two options `-refout` and `-refonly` that control reference assembly generation.

## C# version 7.0

*Released March 2017*

C# version 7.0 was released with Visual Studio 2017. This version has some evolutionary and cool stuff in the vein of C# 6.0. Here are some of the new features:

- Out variables
- [Tuples and deconstruction](#)
- [Pattern matching](#)
- [Local functions](#)
- [Expanded expression bodied members](#)
- [Ref locals](#)
- [Ref returns](#)

Other features included:

- [Discards](#)
- [Binary Literals and Digit Separators](#)
- [Throw expressions](#)

All of these features offer new capabilities for developers and the opportunity to write cleaner code than ever. A highlight is condensing the declaration of variables to use with the `out` keyword and by allowing multiple return values via tuple. .NET Core now

targets any operating system and has its eyes firmly on the cloud and on portability. These new capabilities certainly occupy the language designers' thoughts and time, in addition to coming up with new features.

## C# version 6.0

*Released July 2015*

Version 6.0, released with Visual Studio 2015, released many smaller features that made C# programming more productive. Here are some of them:

- [Static imports](#)
- [Exception filters](#)
- [Auto-property initializers](#)
- [Expression bodied members](#)
- [Null propagator](#)
- [String interpolation](#)
- [nameof operator](#)

Other new features include:

- Index initializers
- Await in catch/finally blocks
- Default values for getter-only properties

If you look at these features together, you see an interesting pattern. In this version, C# started to eliminate language boilerplate to make code more terse and readable. So for fans of clean, simple code, this language version was a huge win.


They did one other thing along with this version, though it's not a traditional language feature in itself. They released [Roslyn the compiler as a service](#)<sup>↗</sup>. The C# compiler is now written in C#, and you can use the compiler as part of your programming efforts.

## C# version 5.0

*Released August 2012*

C# version 5.0, released with Visual Studio 2012, was a focused version of the language. Nearly all of the effort for that version went into another groundbreaking language concept: the `async` and `await` model for asynchronous programming. Here's the major features list:

- [Asynchronous members](#)

- [Caller info attributes](#)
- [Code Project: Caller Info Attributes in C# 5.0](#) 

The caller info attribute lets you easily retrieve information about the context in which you're running without resorting to a ton of boilerplate reflection code. It has many uses in diagnostics and logging tasks.

But `async` and `await` are the real stars of this release. When these features came out in 2012, C# changed the game again by baking asynchrony into the language as a first-class participant.

## C# version 4.0

*Released April 2010*

C# version 4.0, released with Visual Studio 2010, introduced some interesting new features:

- [Dynamic binding](#)
- [Named/optional arguments](#)
- [Generic covariant and contravariant](#)
- [Embedded interop types](#)

Embedded interop types eased the deployment pain of creating COM interop assemblies for your application. Generic covariance and contravariance give you more power to use generics, but they're a bit academic and probably most appreciated by framework and library authors. Named and optional parameters let you eliminate many method overloads and provide convenience. But none of those features are exactly paradigm altering.

The major feature was the introduction of the `dynamic` keyword. The `dynamic` keyword introduced into C# version 4.0 the ability to override the compiler on compile-time typing. By using the `dynamic` keyword, you can create constructs similar to dynamically typed languages like JavaScript. You can create a `dynamic x = "a string"` and then add six to it, leaving it up to the runtime to sort out what should happen next.

Dynamic binding gives you the potential for errors but also great power within the language.

## C# version 3.0

*Released November 2007*

C# version 3.0 came in late 2007, along with Visual Studio 2008, though the full boat of language features would actually come with .NET Framework version 3.5. This version marked a major change in the growth of C#. It established C# as a truly formidable programming language. Let's take a look at some major features in this version:

- [Auto-implemented properties](#)
- [Anonymous types](#)
- [Query expressions](#)
- [Lambda expressions](#)
- [Expression trees](#)
- [Extension methods](#)
- [Implicitly typed local variables](#)
- [Partial methods](#)
- [Object and collection initializers](#)

In retrospect, many of these features seem both inevitable and inseparable. They all fit together strategically. This C# version's killer feature was the query expression, also known as Language-Integrated Query (LINQ).

A more nuanced view examines expression trees, lambda expressions, and anonymous types as the foundation upon which LINQ is constructed. But, in either case, C# 3.0 presented a revolutionary concept. C# 3.0 began to lay the groundwork for turning C# into a hybrid Object-Oriented / Functional language.

Specifically, you could now write SQL-style, declarative queries to perform operations on collections, among other things. Instead of writing a `for` loop to compute the average of a list of integers, you could now do that as simply as `list.Average()`. The combination of query expressions and extension methods made a list of integers a whole lot smarter.

## C# version 2.0

*Released November 2005*

Let's take a look at some major features of C# 2.0, released in 2005, along with Visual Studio 2005:

- [Generics](#)
- [Partial types](#)
- [Anonymous methods](#)
- [Nullable value types](#)
- [Iterators](#)

- [Covariance and contravariance](#)

Other C# 2.0 features added capabilities to existing features:

- Getter/setter separate accessibility
- Method group conversions (delegates)
- Static classes
- Delegate inference

While C# began as a generic Object-Oriented (OO) language, C# version 2.0 changed that in a hurry. With generics, types and methods can operate on an arbitrary type while still retaining type safety. For instance, having a `List<T>` lets you have `List<string>` or `List<int>` and perform type-safe operations on those strings or integers while you iterate through them. Using generics is better than creating a `ListInt` type that derives from `ArrayList` or casting from `Object` for every operation.

C# version 2.0 brought iterators. To put it succinctly, iterators let you examine all the items in a `List` (or other Enumerable types) with a `foreach` loop. Having iterators as a first-class part of the language dramatically enhanced readability of the language and people's ability to reason about the code.

## C# version 1.2

*Released April 2003*

C# version 1.2 shipped with Visual Studio .NET 2003. It contained a few small enhancements to the language. Most notable is that starting with this version, the code generated in a `foreach` loop called `Dispose` on an `IEnumerator` when that `IEnumerator` implemented `IDisposable`.

## C# version 1.0

*Released January 2002*

When you go back and look, C# version 1.0, released with Visual Studio .NET 2002, looked a lot like Java. As [part of its stated design goals for ECMA](#), it sought to be a "simple, modern, general-purpose object-oriented language." At the time, looking like Java meant it achieved those early design goals.

But if you look back on C# 1.0 now, you'd find yourself a little dizzy. It lacked the built-in async capabilities and some of the slick functionality around generics you take for

granted. As a matter of fact, it lacked generics altogether. And [LINQ](#)? Not available yet. Those additions would take some years to come out.

C# version 1.0 looked stripped of features, compared to today. You'd find yourself writing some verbose code. But yet, you have to start somewhere. C# version 1.0 was a viable alternative to Java on the Windows platform.

The major features of C# 1.0 included:

- [Classes](#)
- [Structs](#)
- [Interfaces](#)
- [Events](#)
- [Properties](#)
- [Delegates](#)
- [Operators and expressions](#)
- [Statements](#)
- [Attributes](#)

Article *originally published on the NDepend blog* [↗](#), courtesy of Erik Dietrich and Patrick Smacchia.

#### 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](#)