

Common Language Runtime (CLR) overview

Article • 04/25/2023

.NET provides a run-time environment called the common language runtime that runs the code and provides services that make the development process easier.

Compilers and tools expose the common language runtime's functionality and enable you to write code that benefits from the managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code. Managed code benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

ⓘ Note

Compilers and tools can produce output that the common language runtime can consume because the type system, the format of metadata, and the run-time environment (the virtual execution system) are all defined by a public standard, the ECMA Common Language Infrastructure specification. For more information, see [ECMA C# and Common Language Infrastructure Specifications](#).

To enable the runtime to provide services to managed code, language compilers must emit metadata that describes the types, members, and references in your code. Metadata is stored with the code; every loadable common language runtime portable executable (PE) file contains metadata. The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries.

The runtime automatically handles object layout and manages references to objects, releasing them when they're no longer being used. Objects whose lifetimes are managed in this way are called managed data. Garbage collection eliminates memory leaks and some other common programming errors. If your code is managed, you can use managed, unmanaged, or both managed and unmanaged data in your .NET application. Because language compilers supply their own types, such as primitive types, you might not always know or need to know whether your data is being managed.

The common language runtime makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be tightly integrated. For example, you can define a class and then use a different language to derive a class from your original class or call a method on the original class. You can also pass an instance of a class to a method of a class written in a different language. This cross-language integration is possible because language compilers and tools that target the runtime use a common type system defined by the runtime. They follow the runtime's rules for defining new types and for creating, using, persisting, and binding to types.

As part of their metadata, all managed components carry information about the components and resources they were built against. The runtime uses this information to ensure that your component or application has the specified versions of everything it needs, which makes your code less likely to break because of some unmet dependency. Registration information and state data are no longer stored in the registry, where they can be difficult to establish and maintain. Instead, information about the types you define and their dependencies is stored with the code as metadata. This way, the task of component replication and removal is less complicated.

Language compilers and tools expose the runtime's functionality in ways that are intended to be useful and intuitive to developers. Some features of the runtime might be more noticeable in one environment than in another. How you experience the runtime depends on which language compilers or tools you use. For example, if you're a Visual Basic developer, you might notice that with the common language runtime, the Visual Basic language has more object-oriented features than before. The runtime provides the following benefits:

- Performance improvements.
- The ability to easily use components developed in other languages.
- Extensible types provided by a class library.
- Language features such as inheritance, interfaces, and overloading for object-oriented programming.
- Support for explicit free threading that allows creation of multithreaded and scalable applications.
- Support for structured exception handling.

- Support for custom attributes.
- Garbage collection.
- Use of delegates instead of function pointers for increased type safety and security.
For more information about delegates, see [Common Type System](#).

CLR versions


.NET Core and .NET 5+ releases have a single product version, that is, there's no separate CLR version. For a list of .NET Core versions, see [Download .NET Core](#).

However, the .NET Framework version number doesn't necessarily correspond to the version number of the CLR it includes. For a list of .NET Framework versions and their corresponding CLR versions, see [.NET Framework versions and dependencies](#).

Related articles

[Expand table](#)

Title	Description
Managed Execution Process	Describes the steps required to take advantage of the common language runtime.
Automatic Memory Management	Describes how the garbage collector allocates and releases memory.
Overview of .NET Framework	Describes key .NET Framework concepts, such as the common type system, cross-language interoperability, managed execution, application domains, and assemblies.
Common Type System	Describes how types are declared, used, and managed in the runtime in support of cross-language integration.

 Collaborate with us on
GitHub

The source for this content can
be found on GitHub, where you



.NET feedback

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

can also create and review issues and pull requests. For more information, see [our contributor guide](#).



[Open a documentation issue](#)



[Provide product feedback](#)