**Common Language Runtime Overview**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/ddk909ch(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/ddk909ch(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ddk909ch(VS.85).aspx)

[Microsoft Visual Studio 2010/.NET Framework 4](http://msdn.microsoft.com/en-us/library/ddk909ch(VS.100).aspx)

Compilers and tools expose the runtime's functionality and enable you to write code that benefits from this managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code; it 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.

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 are no longer being used. Objects whose lifetimes are managed in this way are called managed data. Garbage collection eliminates memory leaks as well as some other common programming errors. If your code is managed, you can use managed data, unmanaged data, or both managed and unmanaged data in your .NET Framework 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, and they follow the runtime's rules for defining new types, as well as 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. Rather, information about the types you define (and their dependencies) is stored with the code as metadata, making the tasks of component replication and removal much less complicated.

Language compilers and tools expose the runtime's functionality in ways that are intended to be useful and intuitive to developers. This means that 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 are a Visual Basic developer, you might notice that with the common language runtime, the Visual Basic language has more object-oriented features than before. Following are some benefits of the runtime:

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

If you use Microsoft® Visual C++® .NET, you can write managed code using Visual C++, which provides the benefits of a managed execution environment as well as access to powerful capabilities and expressive data types that you are familiar with. Additional runtime features include:

* Cross-language integration, especially cross-language inheritance.
* Garbage collection, which manages object lifetime so that reference counting is unnecessary.
* Self-describing objects, which make using Interface Definition Language (IDL) unnecessary.
* The ability to compile once and run on any CPU and operating system that supports the runtime.

You can also write managed code using the C# language, which provides the following benefits:

* Complete object-oriented design.
* Very strong type safety.
* A good blend of Visual Basic simplicity and C++ power.
* Garbage collection.
* Syntax and keywords similar to C and C++.
* Use of [delegates](http://msdn.microsoft.com/en-us/library/96b1ayy4.aspx) rather than function pointers for increased type safety and security. Function pointers are available through the use of the **unsafe** C# keyword and the **/unsafe** option of the C# compiler (Csc.exe) for unmanaged code and data.