What is "managed code"?

Article • 04/19/2023

When working with .NET, you'll often encounter the term "managed code". This article explains what *managed code* means and provides additional information around it.

To put it simply, managed code is just that: code whose execution is managed by a runtime. In this case, the runtime in question is called the **Common Language Runtime** or CLR, regardless of the implementation (for example, Mono, .NET Framework, or .NET Core/.NET 5+). The CLR is in charge of taking the managed code, compiling it into machine code and then executing it. On top of that, the runtime provides several important services such as automatic memory management, security boundaries, and type safety.

Contrast this to the way you would run a C/C++ program, also called "unmanaged code". In the unmanaged world, the programmer is in charge of pretty much everything. The actual program is, essentially, a binary that the operating system (OS) loads into memory and starts. Everything else, from memory management to security considerations are a burden of the programmer.

Managed code is written in one of the high-level languages that can be run on top of .NET, such as C#, Visual Basic, F# and others. When you compile code written in those languages with their respective compiler, you don't get machine code. You get **Intermediate Language** code which the runtime then compiles and executes. C++ is the one exception to this rule, as it can also produce native, unmanaged binaries that run on Windows.

Intermediate Language & execution

What is "Intermediate Language" (or IL for short)? It is a product of compilation of code written in high-level .NET languages. Once you compile your code written in one of these languages, you will get a binary that is made out of IL. It is important to note that the IL is independent from any specific language that runs on top of the runtime; there is even a separate specification for it that you can read if you're so inclined.

Once you produce IL from your high-level code, you will most likely want to run it. This is where the CLR takes over and starts the process of **Just-In-Time** compiling, or **JIT-ing** your code from IL to machine code that can actually be run on a CPU. In this way, the CLR knows exactly what your code is doing and can effectively *manage* it.

Intermediate Language is sometimes also called Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL).

Unmanaged code interoperability

Of course, the CLR allows passing the boundaries between managed and unmanaged world, and there's a lot of code that does that, even in the .NET class libraries. This is called *interoperability*, or *interop* for short. These provisions would allow you to, for example, wrap up an unmanaged library and call into it. However, it is important to note that once you do this, when the code passes the boundaries of the runtime, the actual management of the execution is again in the hand of unmanaged code, and thus falls under the same restrictions.

Similar to this, C# is one language that allows you to use unmanaged constructs such as pointers directly in code by utilizing what is known as *unsafe context*, which designates a piece of code for which the execution isn't managed by the CLR.

More resources

- Overview of .NET Framework
- Unsafe Code and Pointers
- Native interoperability