**you compile a C# application**

When you compile a C# application or any application written in a CLS-compliant language, the application is compiled into MSIL. This MSIL is then further compiled into native CPU instructions when the application is executed for the first time by the CLR. (Actually, only the called functions are compiled the first time they are invoked.) However, since we're all geeks here and this book is called *Inside C#*, let's look at what's really happening under the hood:

1. You write source code in C#.
2. You then compile it using the C# compiler (csc.exe) into an EXE.
3. The C# compiler outputs the MSIL code and a manifest into a read-only part of the EXE that has a standard PE (Win32-portable executable) header.

So far, so good. However, here's the important part: when the compiler creates the output, it also imports a function named \_ *CorExeMain* from the .NET runtime.

1. When the application is executed, the operating system loads the PE, as well as any dependent dynamic-link libraries (DLLs), such as the one that exports the \_ *CorExeMain* function (mscoree.dll), just as it does with any valid PE.
2. The operating system loader then jumps to the entry point inside the PE, which is put there by the C# compiler. Once again, this is exactly how any other PE is executed in Windows.

However, since the operating system obviously can't execute the MSIL code, the entry point is just a small stub that jumps to the \_ *CorExeMain* function in mscoree.dll.

1. The \_ *CorExeMain* function starts the execution of the MSIL code that was placed in the PE.
2. Since MSIL code cannot be executed directly—because it's not in a machine-executable format—the CLR compiles the MSIL by using a just-in-time (JIT) compiler (or JITter) into native CPU instructions as it processes the MSIL. JIT compiling occurs only as methods in the program are called. The compiled executable code is cached on the machine and is recompiled only if there's some change to the source code.