Three different JITters

Three different JITters can be used to convert the MSIL into native code, depending on the circumstances:

* **Install-time code generation** Install-time code generation will compile an entire assembly into CPU-specific binary code, just as a C++ compiler does. An assembly is the code package that's sent to the compiler. (I'll talk about assemblies in more detail later in this chapter in "[Deployment](mk:@MSITStore:D:\Ketki_Data\dotnet\doiicd\dnetfundamentals\doc\dotnetInsideCSharp_ARCHER.CHM::/html/32ch02c.htm#uid-0664FCB6-B456-F1A5-1216-F87F3664828F).") This compilation is done at install time, when the end user is least likely to notice that the assembly is being JIT-compiled. The advantage of install-time code generation is that it allows you to compile the entire assembly just once before you run it. Because the entire assembly is compiled, you don't have to worry about intermittent performance issues every time a method in your code is executed the first time. It's like a time-share vacation plan in which you pay for everything up front. While paying for the vacation plan is painful, the advantage is that you never have to worry about paying for accommodations again. When and if you use this utility depends on the size of your specific system and your deployment environment. Typically, if you're going to create an installation application for your system, you should go ahead and use this JITter so that the user has a fully optimized version of the system "out of the box."
* **JIT** The default JITter is called at run time—in the manner I described in the preceding numbered list—each time a method is invoked for the first time. This is akin to a "pay-as-you-go" plan and is the default if you don't explicitly run the PreJIT compiler.
* **EconoJIT** Another run-time JITter, the EconoJIT is specifically designed for systems that have limited resources—for example, handheld devices with small amounts of memory. The major difference between this JITter and the regular JITter is the incorporation of something called *code pitching*. Code pitching allows the EconoJIT to discard the generated, or compiled, code if the system begins to run out of memory. The benefit is that the memory is reclaimed. However, the disadvantage is that if the code being pitched is invoked again, it must be compiled again as though it had never been called.