**Building and Configuring Class Libraries**

During the first four parts of this book, you have created a number of “stand-alone” executable applications, in which all the programming logic was packaged within a single executable file (\*.exe). These executable assemblies were using little more than the primary .NET class library, mscorlib.dll. While some simple .NET programs may be constructed using nothing more than the .NET base class libraries, chances are it will be commonplace for you (or your teammates) to isolate reusable programming logic into custom class libraries (\*.dll files) that can be shared among applications. In this chapter, you will learn about various ways to package your types into custom libraries of code. To begin, you’ll learn the details of partitioning types into .NET namespaces. After this, you will examine the class library project templates of Visual Studio and learn the distinction between private and shared assemblies. Next, you’ll explore exactly how the .NET runtime resolves the location of an assembly, and you’ll come to understand the global assembly cache, XML application configuration files (\*.config files), publisher policy assemblies, and the System.Configuration namespace.

**Defining Custom Namespaces** Before diving into the aspects of library deployment and configuration, the first task is to learn the details of packaging your custom types into .NET namespaces. Up to this point in the text, you’ve been building small test programs that leverage existing namespaces in the .NET universe (System, in particular). However, when you build larger applications with many types, it can be helpful to group your related types into custom namespaces. In C#, this is accomplished using the namespace keyword. Explicitly defining custom namespaces is even more important when creating .NET \*.dll assemblies, as other developers will need to reference the library and import your custom namespaces to use your types.

**Resolving Name Clashes with Aliases** The C# using keyword also lets you create an alias for a type’s fully qualified name. When you do so, you define a token that is substituted for the type’s full name at compile time.