Part I

The .NET Framework

Chapter 1

The .NET Foundation

A rock pile ceases to be a rock pile the moment a single man contemplates it, bearing within him the image of a cathedral.
Saint Exupery, Flight to Arras (1942)

Welcome to XML Web services for ASP.NET. The Web services movement of today has been referred to as a technology that is heralding a new era of application design. You can think of Web services as a component or a method that can be called across the Web. Even though many companies are offering solutions to implement and consume Web services, this book focuses on Microsoft's solution. In this chapter, I will be reviewing the foundation of the .NET Framework and what you need to understand about this new platform in order to see how XML Web services fit into the whole picture. This chapter emphasizes the following topics:

- Grasping the idea of .NET
- Learning about the CLR
- Understanding the Base Class Libraries

What Is .NET?

This is a loaded question, but what it really comes down to is that .NET means different things to different people. Much of what Microsoft is now offering has the .NET name somewhere in its title, but what .NET means really depends on whom you ask. The official one-line answer is that .NET is Microsoft's platform for XML Web services.

Microsoft's .NET Framework is a new computing platform built with the Internet in mind, but without sacrificing the traditional desktop application platform. The Internet has been around for a number of years now, and Microsoft has been busy developing technologies and tools that are totally focused on it. These earlier technologies, however, were built on Windows DNA (Distributed interNet Applications Architecture), which was based on COM (Component Object Model). Microsoft's COM was in development many years before the Internet became the force that we know today. Consequently, the COM model has been built upon and added to in order to adapt it to the changes brought about by the Internet.

With the .NET Framework, Microsoft built everything from the ground up with Internet integration as the goal. Building a platform from the ground up also allowed the .NET Framework developers to look at the problems and limitations that inhibited application development in the past and to provide the solutions that were needed to quickly speed past these barriers.

.NET is a collection of tools, technologies, and languages that all work together in a framework to provide the solutions that are needed to easily build and deploy truly robust enterprise applications. These .NET applications are also able to easily communicate with one another and provide information and application logic, regardless of platforms and languages.

Sounds pretty outstanding, doesn't it?

Figure 1-1 shows an overview of the structure of the .NET Framework.

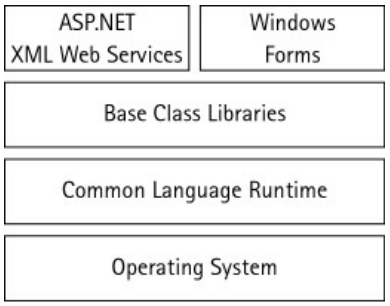

Figure 1-1: The .NET Framework.

The first thing that you should notice when looking at this diagram is that the .NET Framework sits on top of the operating system. Presently, the operating systems that can take the .NET Framework include Windows XP, Windows 2000, and Windows NT. There has also been a lot of talk about .NET being ported over by some third-party companies so that a majority of the .NET Framework could run on other platforms as well.

NOTE: The support for the .NET Framework on Windows NT is limited to functioning only as a client. Windows NT will not support the Framework as a server.

At the base of the .NET Framework is the Common Language Runtime (CLR). The CLR is the engine that manages the execution of the code.

The next layer up is the .NET Framework Base Classes. This layer contains classes, value types, and interfaces that you will use often in your development process. Most notably within the .NET Framework Base Classes is ADO.NET, which provides access to and management of data.

The third layer of the framework is ASP.NET and Windows Forms. ASP.NET should not be viewed as the next version of Active Server Pages after ASP 3.0, but as a dramatically new shift in Web application development. Using ASP.NET, it's now possible to build robust Web applications that are even more functional than Win32 applications of the past. This was always quite difficult to do in the stateless nature of the Internet, but ASP.NET offers a number of different solutions to overcome the traditional limitations on the types of applications that were possible. The ASP.NET section of the .NET Framework is also where the XML Web services model resides.

The second part of the top layer of the .NET Framework is the Windows Forms section. This is where you can build the traditional executable applications that you built with Visual Basic 6.0 in the past. There are some new features here as well, such as a new drawing class and the capability to program these applications in any of the available .NET languages. You may view XML Web services as something you use within the ASP.NET applications that you build, but this isn't the only place where you can consume XML Web services. It's quite possible to consume XML Web services within a Windows Form application and a number of other application types, just as you can from any ASP.NET Web application.

CROSS REFERENCE: ASP.NET and Windows Forms are covered in Chapter 2.

The .NET vision

There is a vision of the future at Microsoft, and this vision is strongly influencing the direction of its products. The vision is that in the future all devices will be connected in one way or another.

The view is that all business and household devices (telephones, microwaves, computers, televisions, and so forth) will be connected to the Internet one day and will, therefore, be able to communicate across this medium to perform the functionality needed to turn them into superior products. The thought is that these products and devices will use standardized languages such as XML and SOAP to communicate over standard protocols such as HTTP. That is where XML Web services come in! XML Web services are going to be the means for all these products and devices to communicate the information and requests that they will need to perform some type of functionality or registration.

Aside from that vision of the future, .NET today is able to solve similar problems in regard to connecting disparate applications and platforms that run on a wide variety of devices. Today there are walls between these differing systems, and .NET has been built to knock down these barriers by providing a common communication model using XML and SOAP.

Microsoft has truly taken hold of this idea and has developed the entire .NET Framework around it, and that is why you will find strong support for XML throughout the Framework. Also, you will notice that when you visit Microsoft's developer pages at MSDN, you cannot get away from stories and articles that are related in one way or another to Web services. This momentum will get stronger and stronger as .NET matures.

It isn't just Microsoft that has grabbed hold of this vision of the future. If you go to IBM's developer site you will notice that all the talk is also about XML Web services and how to use SOAP. Other companies, like Sun Microsystems, have joined in, promoting their own versions of Web services. So you don't need to worry and wonder if the Web services idea is simply a Microsoft fad — it isn't. Web services is a vision that comes from the computing industry and not just one company in particular. Almost every new platform version, database, and server application is being developed with Web services in mind. The companies and people developing these products realize that it is quite advantageous to expose platform or application functionality as Web services.

Imagine a world where all the commercial products and devices you can purchase off the shelf are connected via broadband connection to the Internet. They are thus enabled to trade information and report events with lightning-quick speed. The functionality that can be built upon this type of platform is limitless.

Devices and products of all kinds will be delivering or consuming Web services in this type of environment. In most cases, these Web services will not be developed with just one type of client in mind. Instead, Web services will offer their logic, information, registration service, or whatever they need to offer to a multitude of clients. Figure 1-2 gives you a better idea of what this world of Web services can mean to you.

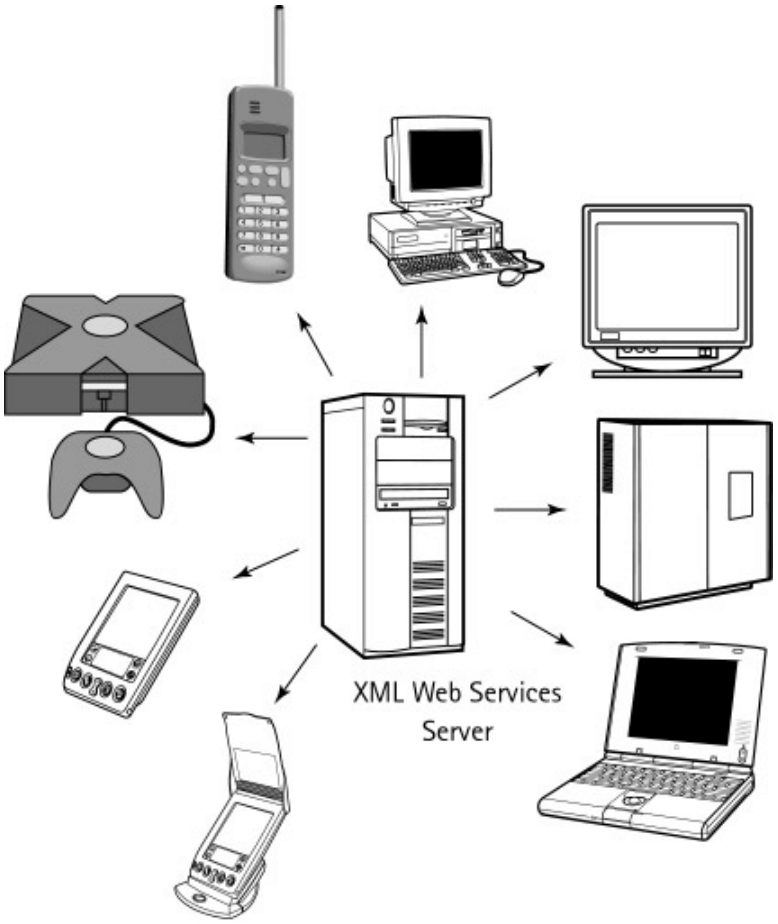

Figure 1-2: One Web service for multiple clients.

XML Web services will deliver these services over standard protocols such as HTTP and SOAP so that any device, regardless of platform, will be able to consume and utilize the

information being sent. You, as a developer, will not have to build a new way to interact with each new device, but instead you can offer a single means for all devices or products to interact with the XML Web service that you will provide.

As a developer, you can see the benefit of an environment where you do not always need to reinvent the wheel but, instead, can procure the functionality and items that you need as you develop your applications or Web sites. You may now be wondering how hard it is to build and consume XML Web services.

Microsoft has realized that, in order for this type of platform to take hold, it has to be simple and easy to use. Development must be uncomplicated, and deployment needs to be a breeze *and this is ex actly w hat the . NE T platform has delivered!*

The .NET solution

In anticipation of this future, Microsoft has developed the .NET Framework and the tools necessary to build in this new environment. Microsoft realized that it had many languages and tools that basically did the same thing. Admittedly, some languages and tools did certain things better than others, but all of them worked towards the same goal, and there was plenty of overlap.

Microsoft has taken the best of all these different worlds and has merged them into a single environment, as shown in Figure 1-3.

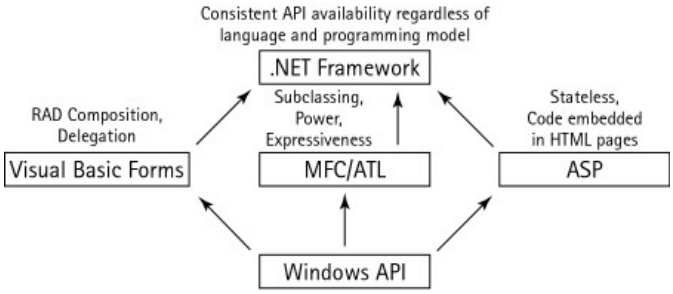

Figure 1-3: A unified model.

This new unified model will allow you, as a developer, to use one development environment and platform to build every application type that you need.

The .NET Framework

To get around the problem of having multiple development environments, Microsoft developed an environment that is a unified development environment. This framework is the platform for the entire .NET idea. The framework is language-neutral and built to provide you with the tools and solutions that you need to build rich applications in the stateless environment of the Internet.

The .NET Framework includes two main components the Common Language Runtime and the Base Class Libraries. Each of these elements is explained later in this chapter.

One of the main objectives of the framework is to provide a simplified development model so that a lot of the plumbing that was required to develop in the past is eliminated. The .NET Framework is a simpler development environment and will give developers more power over their applications. This framework uses the latest in Internet standards such as XML, SOAP, and HTTP. You will also find the applications that you build on this platform are easy to deploy and maintain.

The .NET IDE

Instead of a multitude of tools for development, the .NET team created a single tool for developing Windows Forms, ASP.NET Web Applications, and XML Web services. This tool is Visual Studio .NET.

As a developer, you will find Visual Studio .NET an indispensable tool. It is finely interwoven within the .NET Framework and will give you full and complete access to everything that .NET has to offer. Visual Studio .NET is discussed further in the next chapter.

The .NET languages

In the past, you chose the development language for an application based upon the functionality that you were trying to perform. Some languages were more powerful than others, but at the same time they might have required a higher level of understanding and were, in most cases, more difficult to program in.

Now the .NET Framework provides you with a language-independent programming platform. You do not have to decide which language would provide a better solution. All languages are now on a level playing field.

In .NET, no one language is superior to any of the other languages. They all have equal access to everything that .NET offers.

To be part of the .NET Framework, a language only has to follow certain rules. The biggest and most important rule for inclusion is that the language needs to be an object-oriented language. Microsoft provides four languages with the .NET Framework: Visual Basic .NET, C#, C++ .NET, and JScript .NET. Microsoft also provides J# (pronounced J-sharp), but in order to use this new language that is basically Java for .NET, you need to download the language to install it on your server. You can find this download at `http://msdn.microsoft.com`.

The .NET Enterprise Servers

The marketing folks at Microsoft made things a little more confusing when they started naming everything with a .NET suffix. Shortly after the introduction of .NET, all the forthcoming .NET Enterprise Servers rebranded themselves as part of the .NET platform.

The problem is that they *really* aren't part of the .NET platform at all. Yes, it is true that you can interact with some of these applications in an indirect way through the framework, but the first *true* .NET server will be the next version of Microsoft SQL Server following SQL Server 2000. There has been no indication of how this next version will be folded into the .NET platform. However, it has been stated that these servers will be deeply integrated and that, for instance, you will be able to write your stored procedures in any of the .NET languages.

This current set of .NET Enterprise Servers is considered part of the .NET platform mainly because they are tightly coupled with XML, a language the .NET platform understands very well, thus allowing them to communicate and work together with ease.

.NETs Foundation

The foundation of the .NET platform is the .NET Framework, which we have already introduced. The .NET Framework sits on top of the operating system and is made up of two parts, the Common Language Runtime and the Base Class Libraries. Each one of these parts plays an important role in the development of .NET applications and services.

The Common Language Runtime

Many different languages and platforms provide a runtime, and the .NET Framework is no exception. You will find, however, that this runtime is quite different from most.

The Common Language Runtime (CLR) in the .NET Framework manages the execution of the code and provides access to a variety of services that will make the development process easier.

The CLR has been developed to be far superior to previous runtimes, such as the VB runtime, by attaining the following:

- Cross-language integration
- Code access security
- Object lifetime management
- Debugging and profiling support

Code that is compiled and targeted to the CLR is known as *managed code*. Managed code provides *metadata* that is needed for the CLR to provide the services of multilanguage support, code security, object lifetime management, and memory management.

NOTE: *Metadata* is basically "data about data" or a description of the contents of a .NET component. This metadata is stored within the assembly manifest. In the past, it was difficult for components written in competing languages to interact with one another. The .NET Framework uses metadata so that .NET components are self-describing, making them easy to interoperate with other components.

Compilation to managed code

The .NET Framework requires that you use a language compiler that is targeted at the CLR, such as the Visual Basic .NET, C#, C++ .NET, or JScript .NET compilers provided by Microsoft. Note that there are a lot of third-party compilers on the market (such as COBOL and Perl), and the number will continue to grow.

So how does the code that you typed into Visual Studio .NET become the code that the user receives when he is using your application? It is fairly simple and straightforward. Figure 1-4 shows a diagram of the compilation process.

After using one of the language compilers, your code is compiled down to *Microsoft Intermediate Language*Microsoft Intermediate Language, known as MSIL or simply IL, is a CPU-independent set of instructions that can be easily converted to native code. The metadata is also contained within the IL.

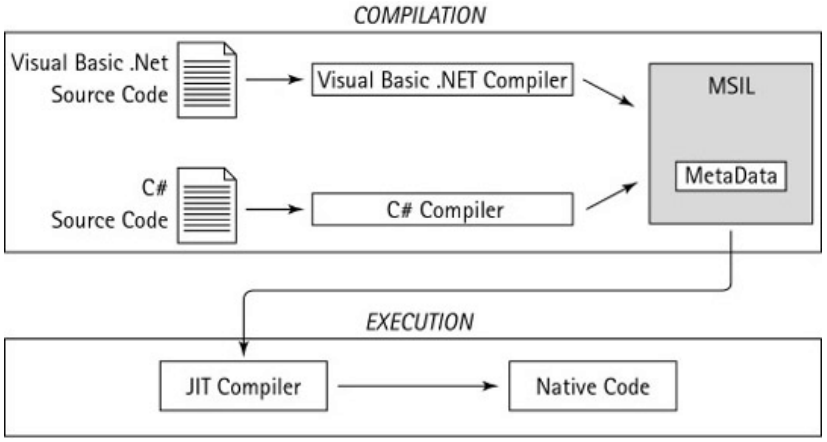

Figure 1-4: Managed code execution process.

The IL is CPU-independent. This means that IL code is not reliant on the specific computer that generated it. In other words, it can be moved from one computer to another (as long as the computer supports the .NET Framework) without any complications. This is what makes X-Copy, or just copying over the application, possible.

After IL, the code that you started with will be compiled down even further by the JIT compiler to *machine code* or *native code*. The IL contains everything that is needed to do this, such as the instructions to load and call methods and a number of other operations.

Jit compilation

The .NET Framework contains one or more JIT compilers that compile your IL code down to machine code, or code that is CPU-specific. This is done when the application is executed for the first time.

You will notice this process after you build your first ASP.NET page. After you build any ASP.NET page, you compile the page down to IL. When you go to the browser and call the page by typing its URL in the address bar, you notice a slight pause of a few seconds as the computer seems to think about what it is doing. It is actually calling this IL code and converting it with a JIT compiler to machine code. This happens only the first time that someone requests the page. After the first time, you can hit F5 to refresh the page, and the page is immediately executed. The page has already been converted to machine code and is now stored in memory. The CLR knows the JIT compiler has already compiled the page. Therefore, it gets the output of the page from memory. If you later make a change to your ASP.NET page, recompile, and then run the page again, CLR detects that there was a change

to the original file. It uses the JIT compiler once again to compile the IL code down to machine code.

The JIT compiler, as it compiles to machine code, makes sure that the code is type safe. It does this to ensure that objects are separate, thereby making certain that objects won't unintentionally corrupt one another.

Assemblies

In the applications that you build within the .NET Framework, assemblies will always play an important role. Assemblies can be thought of as the building blocks of your applications. Without an associated assembly, code will not be able to compile from IL. When you are using the JIT compiler to compile your code from managed code to machine code, the JIT compiler will look for the IL code that is stored in a portable executable (PE) file along with the associated assembly manifest.

Every time you build a Web Form or Windows Form application in .NET, you are actually building an assembly. Every one of these applications will contain at least one assembly.

As in the Windows DNA world where DLLs and EXEs are the building blocks of applications, in the .NET world, it is the assembly that is the used as the foundation of applications.

In the world of Windows DNA and COM, there was a situation that was referred to as DLL Hell. COM components were generally designed so that there was only one version of that COM component on a machine at any given time. This was because the COM specification did not provide for the inclusion of dependency information in a component's type definition. When the developer had to make some changes to the COM component, this new component was introduced and, in many cases, broke applications.

With .NET, it is now possible to have multiple versions of components, or assemblies, running on the same server side by side. An application will always look for the assembly that built it.

When an application is started in .NET, the application will look for an assembly in the installation folder. Assemblies that are stored in a local installation folder are referred to as *private assemblies*. If the application cannot find the assembly within the installation folder, the application will turn to the *Global A ssembly Cache (GA C)*for the assembly.

The GAC is a place where you can store assemblies that you want to share across applications. You can find the assemblies that are stored in the GAC in the `WINNT\ASSEMBLY` folder in your local disk drive.

The structure of an assembly

Assemblies contain code that is executed by the Common Language Runtime. The great thing about assemblies is that they are self-describing. All the details about the assembly are stored within the assembly itself. In the Windows DNA world, COM stored all its self-describing data within the server's registry, and so installing (as well as uninstalling) COM components meant shutting down IIS. Because .a NET assembly stores this information within itself, it makes XCOPY functionality possible. Installing an assembly is as simple as copying it, and there is no need to stop or start IIS while this is going on.

Figure 1-5 shows the structure of an assembly.

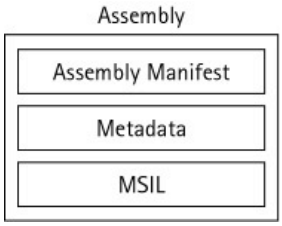

Figure 1-5: A diagram of an assembly.

Assemblies are made up of the following parts:

- The assembly manifest
- Type metadata
- Microsoft Intermediate Language (MSIL) code

The assembly manifest is where the details of the assembly are stored. The assembly is stored within the DLL or EXE itself. Assemblies can either be single or multifile assemblies and, therefore, assembly manifests can either be stored in the assembly or as a separate file. The assembly manifest also stores the version number of the assembly to ensure that the application always uses the correct version. When you are going to have multiple versions of an assembly on the same machine, it is important to label them carefully so that the CLR knows which one to use. Version numbers in assemblies are constructed in the following manner:

```
<major version>.<minor version>.<build number>.<revision>
```

Type metadata was explained earlier in this chapter as data about data. This metadata contains information on the types that are exposed by the assembly such as security permission information, class and interface information, and other assembly information.

Garbage collection

The .NET Framework is a garbage-collected environment. Garbage collection is the process of detecting when objects are no longer in use and automatically destroying those objects, thus freeing memory.

Garbage collection is not a new concept. It has been used in other languages for quite some time. In fact, Java has a garbage collection system in place. Other languages, such as C++, do not have a garbage collection. C++ developers themselves are required to take care of the destruction of objects and the freeing of memory. This results in a number of problems, such as memory leaks. If the developer forgets to free objects from the application, memory allocation of the application grows, sometimes substantially. Also, freeing objects too early causes application bugs to crop up; these kinds of errors are, in most cases, quite difficult to track down.

In .NET, this new garbage collector works so that you as a developer are no longer required to monitor your code for unneeded objects and destroy them. The garbage collector will take care of all this for you. Garbage collection does not happen immediately, but instead the garbage collector will occasionally make a sweep of the heap to determine which objects should be allocated for destruction. This new system completely absolves the developer from hunting down memory usage and deciding when to free memory.

With this new garbage collector, you can control certain aspects of its functions, as it works behind the scenes in your application. Within the SDK documentation, look under the `System.GC` class for more information.

Namespaces

The .NET Framework is made up of hundreds of classes. Many of the applications that you build in .NET are going to take advantage of these classes in one way or another. Because the number of classes is so large and you will need to get at them in a logical fashion, the .NET Framework organizes these classes into a class structure called a *namespace*. There are a number of namespaces, and they are organized in an understandable and straightforward way.

NOTE: `System` is the base namespace in the .NET Framework. All namespaces that are provided in the framework start at this base namespace. For instance, the classes that deal with data access and manipulation are found in the `System.Data` namespace. Other examples include `System.IO`, `System.XML`, `System.Collections`, `System.Drawing`, and so forth. In the naming conventions of namespaces, `System.XML.XMLReader` represents the `XMLReader` type, which belongs to the `System.XML` namespace.

You can import a namespace into your application in the following manner:

VB

```
Imports System.Data
```

C#

```
using System.Data;
```

If you are going to import more than one namespace into your application, do so as shown in the following code:

VB

```
Imports System.Data
Imports System.Data.OleDb
```

C#

```
using System.Data;
using System.Data.OleDb;
```

By importing a namespace into your application, you no longer need to fully qualify the class. For example, if you do not import the namespace into your application, you must write out the full class name.

VB

```
myConnection = "Initial Catalog=Northwind;Data Source=localhost;
                Integrated Security=SSPI;"
Dim conn As New System.Data.SQLClient.SqlConnection(myConnection)
```

C#

```
myConnection = "Initial Catalog=Northwind;Data Source=localhost;
                Integrated Security=SSPI;";
SqlConnection conn = new
System.Data.SQLClient.SqlConnection(myConnection);
```

If you do import the `System.Data.SQLClient` namespace into your application or to the page where you need it, you can refer to the class quite simply, as shown in the following code:

VB

```
myConnection = "Initial Catalog=Northwind;Data Source=localhost;
                Integrated Security=SSPI;"
Dim conn As New SqlConnection(myConnection)
```

C#

```
myConnection = "Initial Catalog=Northwind;Data Source=localhost;
                Integrated Security=SSPI;";
SqlConnection conn = new SqlConnection(myConnection);
```

You can import a namespace directly to an `.aspx` page inline in the following manner for both Visual Basic .NET and C#:

VB and C#

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

Notice in the preceding code that because you imported two namespaces into the page inline, you had to put each of the namespaces within its own set of opening and closing brackets.

When building your .NET applications in the .NET Framework, a number of namespaces are already automatically imported into your application for use throughout. An ASP.NET Web application automatically imports the following namespaces:

```
System
System.Data
System.Drawing
System.Web
System.XML
```

You can find a list of all the imported namespaces in the References folder in the application root. A Windows Form application automatically imports the following namespaces:

```
System
System.Data
System.Drawing
System.Windows.Forms
System.XML
```

An ASP.NET Web Service automatically imports the following namespaces:

```
System
System.Data
System.Web
System.Web.Services
System.XML
```

It is possible to import additional namespaces for use application-wide by right-clicking on the References folder and selecting Add Reference. You will then be able to import any of the other namespaces that are available.

The Base Class Libraries

The Base Class Libraries (BCL) are a set of classes, value types, and interfaces that give you access to specific developer utilities and various system functions. The preceding section covered how these classes are organized by the system of namespaces that are in place in the .NET Framework. Now take a closer look at the classes that are provided in the .NET Framework.

All the languages that sit on top of the .NET Framework have equal access to all the classes that are contained within the BCL. This actually makes it quite easy to look at code from another .NET language and understand what is going on in the code. You use the `SQLConnection` class in Visual Basic .NET to connect to SQL Server exactly the way you do in C#, although the language semantics are different.

Table 1-1 gives a brief description of some of the classes available with the .NET Framework. This table is not a comprehensive list of available namespaces. You can find all the namespaces within the .NET SDK documentation. It is important to realize that you are not limited only to these namespaces. You can create your own namespaces to use within your applications, and there will be third-party namespaces on the market that are available to use as well. One example of a third-party namespace is the `System.Data.Oracle` namespace.

Table 1 - 1 Namespace Definitions

Namespace	*Description*
`System`	Provides base data types and almost 100 classes that deal with situations like exception handling, mathematical functions, and garbage collection.
`System.CodeDom`	Provides the classes needed to produce source files in all the .NET languages.

Namespace	*Description*
`System.Collections`	Provides access to collection classes such as lists, queues, bit arrays, hash tables, and dictionaries.
`System.ComponentModel`	Provides classes that are used to implement runtime and design-time behaviors of components and controls.
`System.Configuration`	Provides classes and interfaces that allow you to programmatically access the various configuration files that are on your system, such as the web.config and the machine.config files.
`System.Data`	Provides classes that allow data access and manipulation to SQL Server and OleDb data sources. These classes make up the ADO.NET architecture.
`System.Diagnostics`	Provides classes that allow you to debug and trace your application. There are classes to interact with event logs, performance counters, and system processes.
`System.DirectoryServices`	Provides classes that allow you to access Active Directory.
`System.Drawing`	Provides classes that allow you to access the basic and advanced features of the new GDI+ graphics functionality.
`System.EnterpriseServices`	Provides classes that allow you to access COM+ services.
`System.Globalization`	Provides classes that access the global system variables, such as calendar display, date and time settings, and currency display settings.
`System.IO`	Provides classes that allow access to file and stream control and manipulation.
`System.Management`	Provides access to a collection of management information and events about the system, devices, and applications designed for the Windows Management Instrumentation (WMI) infrastructure.
`System.Messaging`	Provides classes that allow you to access message queue controls and manipulators.
`System.Net`	Provides access to classes that control network services. These classes also allow control over the systems sockets.
`System.Reflection`	Provides classes that allow control to create and invoke loaded types, methods, and fields.
`System.Resources`	Provides classes that allow you to create and

Namespace	*Description*
	manage culture-specif ic resources.
`System.Runtime.Remoting`	Provides classes that allow the management of remote objects in a distributed environment.
`System.Security`	Provides classes that allow access to authentication, authoriz ation, cryptography, permissions, and policies.
`System.ServiceProcess`	Provides classes that give control over Windows services.
`System.Text`	Provides classes f or working with and manipulating tex t strings.
`System.Threading`	Provides classes f or threading issues and allows you to create multithreaded applications.
`System.Timers`	Provides the capability to raise events on specif ied intervals.
`System.Web`	Provides numerous classes that are used in ASP.NET Web application development.
`System.Web.Services`	Provides classes that are used throughout this book to build, deploy, and consume Web services.
`System.Windows.Forms`	Provides classes to build and deploy Windows F orms applications.
`System.Xml`	Provides classes to work with and manipulate XML data.

As you can tell from the preceding table, many classes are at your disposal for building rich .NET applications. With these classes, much of the plumbing that you had to deal with in the past is now taken care of for you. This book will touch on a number of these classes as you build your XML Web services. With such a large number of classes at your disposal, you may find yourself searching high and low for a certain class. There are a few ways that you can look for the class that you are trying to find. The first is to try the .NET Framework SDK documentation. Another option is to use the Windows Forms Class Viewer, or the WinCV tool. This tool is provided with the .NET Framework. The WinCV tool enables you to quickly look up information on classes in the CLR based upon your custom search criteria. The WinCV tool displays information by reflecting on the type using the CLR reflection API.

To find the `WinCV.exe` tool, look in the `C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin` directory. The left-hand pane of the WinCV tool shows your search results, and the right-hand pane shows the type definition. The type definition is shown in a C#-like syntax (Figure 1-6).

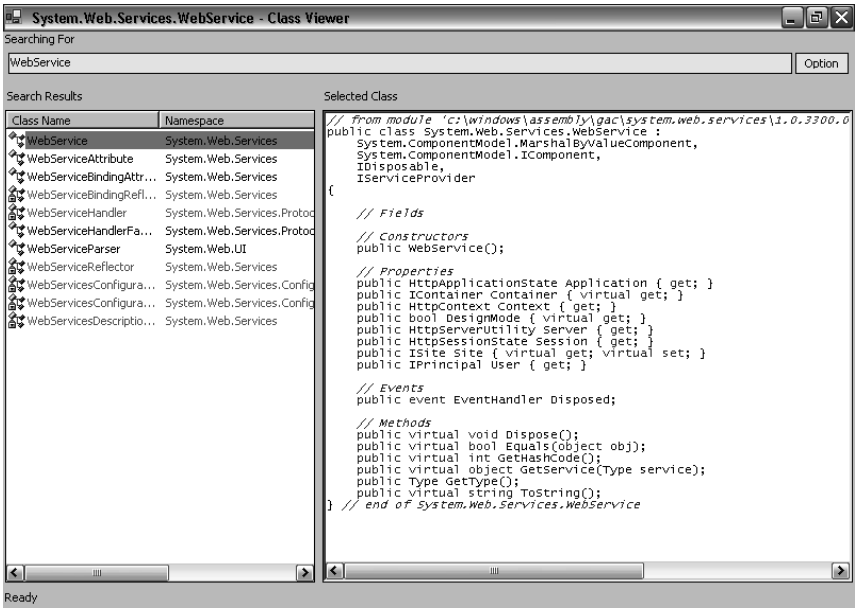

Figure 1-6: Looking at the WebService class with the Windows Forms Class Viewer.

To see the WinCV in action, type **`WebService`** into the search box. You are presented with a large list of classes in the left-hand pane. Clicking on the first choice, WebService, you are shown the type definition for the WebService class in the right-hand pane. The beginning of the definition tells you that the WebService class is part of the `System.Web.Services` namespace. This is a useful tool for search purposes and for a better understanding of classes within the .NET Framework.

Summary

This chapter provides a quick overview of the .NET Framework and gives you a better understanding of what you will use to build your XML Web services. This chapter was one of those 40,000-feet views of the framework, but it provides enough information for you to learn what you need to know to build XML Web services. If you are looking for more information, there are numerous books out there that focus primarily on the .NET Framework.

In the next chapter you look at some of the languages that you will use to build your XML Web services and how XML Web services interact with the .NET Framework. Now, take a closer look at the tools and technologies that you must understand to build XML Web services.