Chapter 1

Introducing .NET and ASP.NET

by Bill Evjen

If you are new to .NET—*Welcome!*

If you are a .NET Framework 1.0 veteran—*Welcome to .NET version 1.1!*

.NET 1.0 was introduced with tremendous excitement. It was original, answered many developers' problems, and truly leap-frogged any of the other technologies out there—especially in the realm of browser-based Internet application development—or ASP.NET.

.NET 1.1 and ASP.NET 1.1 are minor releases and should not be considered substantially different versions from 1.0. The next major revision of .NET and ASP.NET will come with the release of version 2.0.

This chapter introduces the .NET Framework 1.1 and also shows you what's new in ASP.NET 1.1.

Welcome to .NET

Every so often, a technology company needs to step back from itself, look at what it is trying to achieve, and then determine whether it needs to try a different approach. Microsoft did this as it stood back from the COM world it had created and asked itself, "Is there a better way?" The .NET Framework is this better way.

.NET is confusing to many people because the Microsoft marketing folks took hold of the name and started applying it to every product that Microsoft produced. .NET had almost nothing to do with many products that acquired the .NET moniker. This problem is slowly being corrected. More products are now coming from Microsoft without *.NET* in their titles.

Microsoft's short definition of .NET is *Microsoft's platform for building XML Web services.* But the *.NET* that I am talking about (and the .NET that you should be focused on understanding) is the *.NET Framework* itself.

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 Distributed InterNet Applications Architecture (DNA), which was based on the Component Object Model (COM). 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 world of the Internet.

The .NET Framework enabled Microsoft to build everything from the ground up with the Internet in mind. Therefore, the .NET Framework focuses heavily on Internet-enabling your applications, whether these applications are thin-client or thick-client applications. Building a new platform from the ground up also allowed Microsoft to take a close look at how developers developed. Even more importantly, Microsoft began examining how to correct the problems developers experience and how to make them more productive in this new environment.

.NET is a collection of tools, technologies, and languages that all work together in a framework to provide the solutions necessary for easily building and deploying truly robust enterprise applications. These .NET applications can also communicate with one another and provide information and application logic, regardless of platforms and languages.

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

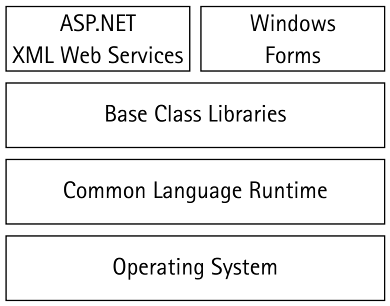


Figure 1-1: The .NET Framework.

You might have seen this simple diagram of the .NET Framework before, but take a closer look at it now. Consider the .NET Framework as something that sits on an operating system. Presently, the operating systems that can take the .NET Framework include Windows Server 2003, Windows XP Professional, Windows 2000, and Windows NT.

note **Support for the .NET Framework on Windows NT is limited to functioning as a client. Windows NT does not support the Framework as a server.**

At the bottom layer of the .NET Framework is the Common Language Runtime (CLR). The CLR is the engine that manages the execution of the code, takes care of object management, provides security, and so much more.

The next layer up from the CLR is the .NET Framework Base Class Libraries (BCL). The BCL layer contains classes, value types, and interfaces that are often used in the development process. The many classes in the BCL are organized into a series of namespaces.

The third layer of the .NET Framework includes the Windows Forms model (more on that shortly), along with the area that is the primary focus of this book—ASP.NET. Don't think of ASP.NET as the latest version of Active Server Pages—the one that comes after ASP 3.0. Instead, think of it as a dramatic 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, a difficult feat because of the stateless nature of the Internet. ASP.NET offers a number of different solutions to overcome the traditional limitations on the types of possible applications. The ASP.NET section of the .NET Framework is also where the XML Web services model resides.

As I mentioned a moment ago, ASP.NET shares the top layer of the .NET Framework with Windows Forms applications. These are the traditional .exe applications, or Win32 thick-client applications. The programming models of the ASP.NET world and the Windows Forms world are quite similar, and these two models use the same objects from the Framework to accomplish their tasks. If you become a good ASP.NET programmer, you also become a good Windows Forms programmer. Interestingly enough, because the models are so similar, you can build a class that you can use in either your ASP.NET application or in a Windows Forms application.

Looking to the future

The .NET Framework is focused on how people will use technology in the future. One view (held not only by Microsoft) says that all devices and applications will one day be connected to the Internet. When this day arrives, people won't use the Internet just for the browser-based applications as they currently do, but also for telephones, televisions, microwaves, Xbox and PlayStation units, and much more (see Figure 1-2). All these devices connected by the Internet can't possibly be built on a single platform. The world will continue to have multiple platforms and languages at its disposal for a long time to come. Consequently, we need a common language that can communicate with the various platforms, applications, and devices.

XML and SOAP are the common languages now facilitating communication among platforms, applications, and devices. By using these languages, devices can communicate over the Internet in an easy and straightforward manner.

Microsoft enables you to start bringing this disparate but connected vision to reality with the .NET Framework. By building XML Web services, you are expressing objects and using XML and SOAP to communicate the object's data or the application logic that it might provide.

cross ref

XML Web services are considered part of the ASP.NET model and are covered in Part VI of this book.

You can also use your ASP.NET applications to consume data or application logic from other Web services on the Internet, regardless of the language or platform used in the creation of these Web services. Over time, more and more connections of applications with remote objects will use the XML Web services model in the .NET Framework.

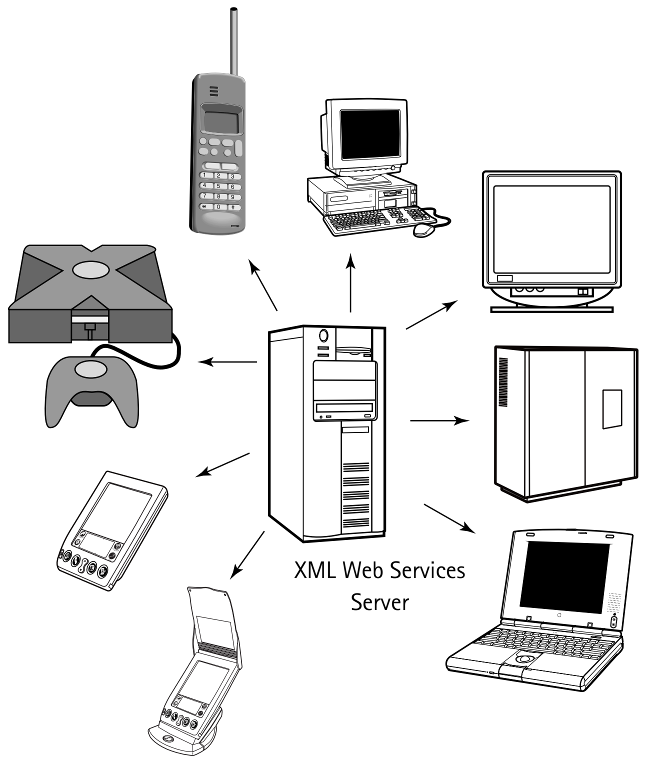


Figure 1-2: One Web service for multiple clients.

Microsoft's .NET solution

Because of the prospect that the universe of devices and applications will be placed on the same medium and will be able to communicate with one another using a common language, Microsoft developed the .NET Framework. Developers using this platform can build their applications to take advantage now of this vision of the future.

Before the .NET Framework came along in 2002, Microsoft offered quite a number of tools and technologies for building a wide variety of applications. As a developer, you could choose the environment, language, or tool appropriate for what you were trying to accomplish.

For example, if you wanted to quickly build a thick-client application, you used Visual Basic. If you wanted to build low-level applications that gave you granular access to the platform, you used C++. If you wanted to build browser-based applications, you used Active Server Pages (ASP).

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

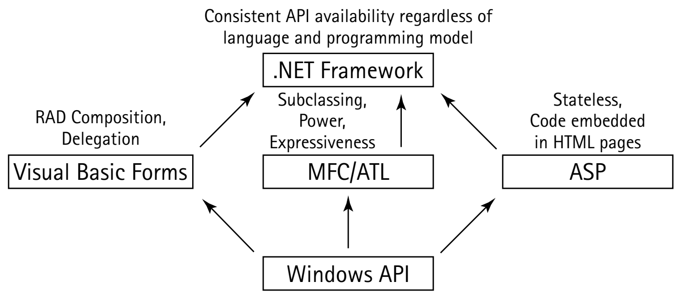

Consistent API availability regardless of language and programming model
.NET Framework
RAD Composition, Delegation
Subclassing, Power, Expressiveness
Stateless, Code embedded in HTML pages
Visual Basic Forms
MFC/ATL
ASP
Windows API

Figure 1-3: A unified model.

Because of this new unified model, you need only one environment, one platform, and any .NET language to build any .NET application. It doesn't matter if the application is a desktop application, a browser-based application, a component, or even a driver. You now have a single environment to do all this work.

One of the main objectives of the Framework is to provide a simplified development model that eliminates a lot of the plumbing required to develop in the past. The .NET Framework gives developers more power over their applications. This framework uses the latest in Internet standards such as XML, SOAP, and HTTP. The applications that you build on this platform are easier to deploy and maintain.

.NET-capable IDEs

Instead of multiple development environments for building various types of applications, Microsoft provides a single development environment to build any type of .NET application. This development environment is Visual Studio .NET. It's the development environment that I recommend you use with this book as you build and work with your .NET applications.

Visual Studio .NET enables you to build your ASP.NET applications in a code-only manner (writing code directly) or by dragging and dropping objects onto a design surface. You can also use Visual Studio .NET to build thick-client applications or Windows Forms and to build your classes, components, and everything else you need.

Another development environment offered by Microsoft is the ASP.NET Web Matrix. This free tool is available from the ASP.NET Web site at `www.asp.net`. It is a great tool to help you get a feel for building Web applications. Unlike Visual Studio .NET, however, the ASP.NET Web Matrix limits you to building only ASP.NET applications and XML Web services. You can't build any Windows Forms applications using the Web Matrix.

cross ref

You learn about the Visual Studio .NET environment in Chapter 2 and the ASP.NET Web Matrix environment in Chapter 3.

note

Besides these two Microsoft IDEs, other IDEs are capable of building .NET applications. One of the more popular development environments is Macromedia's Dreamweaver MX. This IDE enables you to build ASP.NET applications as well as many other types, such as PHP and ColdFusion applications.

You can also build ASP.NET applications by using Notepad and the compilers provided with the .NET Framework. In fact, you can type much of the code from this book into Notepad to build your ASP.NET applications.

The .NET languages

When developing on the Microsoft platform in the past, you chose the development language based on the type of application you wanted to build.

The .NET Framework has completely changed this scenario for us. The .NET Framework enables you to build any type of application or object that can be used in the .NET platform, and you can also use any of the .NET languages to build these applications.

For a language to be part of the .NET Framework, a language has to follow certain rules. The biggest and most important rule for inclusion is that the language must be an object-oriented language. Microsoft provides five languages with the .NET Framework: Visual Basic .NET, C#, C++.NET, JScript .NET, and now J#.

You now have a choice of which language you are going to use to build your ASP.NET applications. For instance, you are not limited to just using Visual Basic .NET to build your browser-based applications; you can also use C#, JScript .NET, or even J# to build a Web-based application.

Whenever this idea is presented, people always ask: *Which is the better language to use? Does one language provide better performance than another?*

The answer to both these questions is an emphatic *NO*. There isn't one language that is better than another. They all have access to the same classes and capabilities that are provided with the .NET Framework. If there is no difference in which language you can use to build your .NET applications, it then becomes simply a matter of style when deciding which of the languages to use.

In addition to the languages that are provided with the install of the .NET Framework 1.1, quite a number (more than 40) of third-party languages are capable of being used in the .NET platform to build your .NET applications. For instance, you can also build ASP.NET applications using COBAL or Perl. To use either of these applications, however, you must install the language yourself because neither is provided with the default install of the .NET Framework.

This book focuses on the two most popular .NET languages: Visual Basic .NET and C#.

A Closer Look at the .NET Foundation

The foundation of the .NET platform is the .NET Framework, which I introduced in the preceding section. 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 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. Its runtime, however, 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 make the development process easier.

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

- 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 has the metadata 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

When creating a .NET application, use one of the .NET languages. After the code is written, you (or Visual Studio .NET) uses the language's compiler to compile the code down to Microsoft Intermediate Language (IL). IL is a CPU-independent set of instructions that can easily be converted to native code. The metadata (the self-describing information about the object created) is also contained within the IL. This is illustrated in Figure 1-4. You can see from the diagram of the compilation process that it really doesn't matter which language you are using with the .NET platform. In the end, each .NET language compiles down to the same type of object.

The IL is CPU-independent, which means that the object created by the compiler is not dependent upon the computer that created it. It, therefore, can be moved at this stage from the computer that created it to any other computer that has the .NET Framework on it. And because this IL object is self-describing, it doesn't need to be registered in the Windows registry. You only have to copy it to the hard drive of the computer—that's it. It makes XCopy possible with .NET applications.

To run the application from here, it actually needs to be compiled even further—down to native code. This is handled by the .NET Framework's JIT compiler. The IL contains everything that is needed to do this, such as the instructions to load, call methods, and a number of other operations.

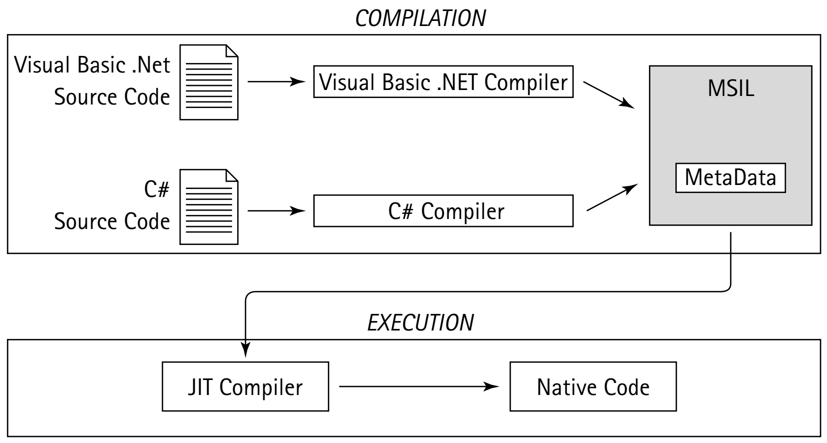


Figure 1-4: Managed-code execution process.

Compiling further with JIT compilation

When the ASP.NET application is invoked for the first time, one of the .NET Framework's JIT compilers takes the IL code and compiles it down to native code. The native code or machine code that is produced is CPU-specific code.

This compilation process is quite apparent when working with ASP.NET. You can build a page in ASP.NET and compile it down to IL using Visual Studio .NET. When you invoke the page by typing the endpoint of the Web page in the browser (using the page's URL), it is run through a JIT compiler. The JIT compiler used depends upon the architecture of the server hosting the `.aspx` page. After the page is compiled down to native code, the code is executed, and the page class that is created outputs the appropriate code to the end user's browser.

This compilation process takes time, and you can see this time elapse when you first invoke an ASP.NET page. There is a pause of a few seconds before the page is generated. After the JIT compilation is complete, however, the page has been compiled into CPU-specific code, and this process doesn't need to occur again. So when a second end user invokes the same ASP.NET page, that user doesn't have to wait for the JIT compilation to take place again. The page output is quick. You can see this by simply pressing F5 (for Refresh). Notice how quickly the page is retrieved.

If you make a change to the code of the ASP.NET page, the page goes through the entire compilation process again. If you make a change to the code and re-invoke the ASP.NET, you notice the same slow start, but only with the first invocation.

Also the JIT compiler, as it compiles to machine code, ensures that the code is type safe. It makes certain that objects are separate, thereby ensuring that these objects won't unintentionally corrupt one another.

An interesting point to understand with JIT compilation and ASP.NET applications is that the entire ASP.NET application is not JIT-compiled when only one of the `.aspx` pages of the application is invoked. Instead, the JIT compiler compiles only the code that is needed by the application at the point of invocation. The pieces of your application that are not needed at the point of invocation are not be compiled down to native code until they are called for the first time. The more your ASP.NET application is used, therefore, the faster it runs.

Assemblies: The .NET building blocks

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

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

One of the big problems that assemblies take care of for .NET developers is solving the problem of "DLL Hell." The world of Windows DNA and COM ensured that only one version of a component existed on a computer at any given time. 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.

The components generated by .NET are self-describing as mentioned earlier in the chapter. This means that the component need not be registered in the registry. Components can be run directly from the ASP.NET application's root directory. Assemblies that are run from the assembly from the local application folder are referred to as *private assemblies*. These assemblies can be copied from one computer to another without any problem. When an ASP.NET application is invoked, it first looks for the needed assembly (a private assembly) in the local directory where the application resides. If none is found in the application itself, the application looks in the Global Assembly Cache (GAC).

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 `WINDOWS\assembly` folder in your local disk drive. Assemblies that are stored within the GAC are referred to as *global assemblies*.

The .NET Framework Class Library: BCL

Developers coming from the Active Server Pages 3.0 world to the .NET world find that they don't need to build as much application plumbing (required in the ASP 3.0 days) because a large number of classes are available with the .NET Framework.

ASP 3.0 had just a handful of classes, such as the `Request` and `Response` objects, which took care of some tasks for the developer. Now with ASP.NET, you have a huge number of classes at your disposal. These classes are part of the default .NET Framework and are referred to as the Base Class Libraries (BCL).

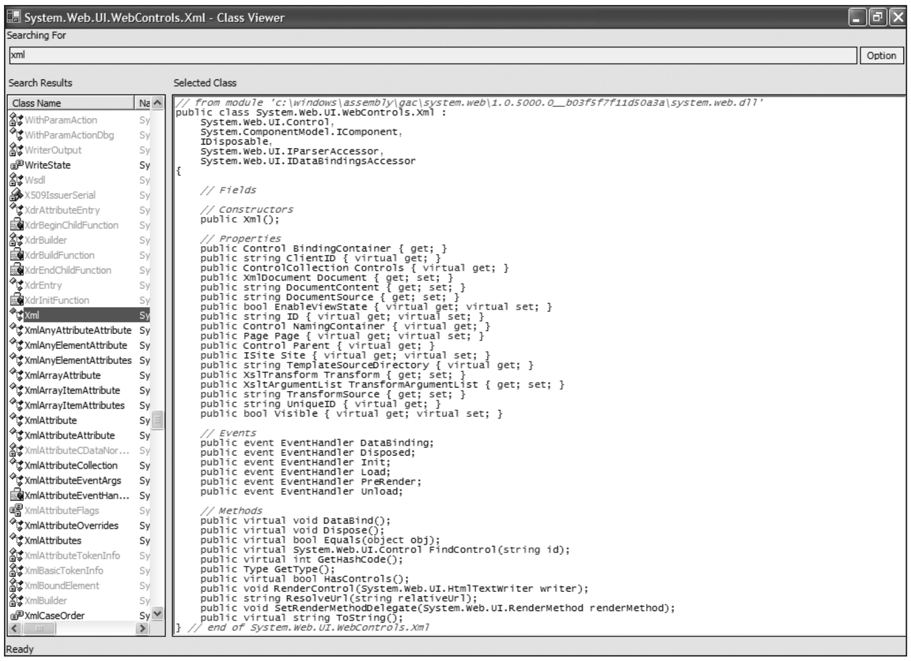
The *Base Class Libraries* provided with the .NET Framework are a set of classes, value types, and interfaces that give you access to specific developer utilities and various system functions. These classes are organized through a system of namespaces.

The great thing about .NET is that all the .NET languages have equal access to all the classes that are available in the framework. One language doesn't have a higher level of access to the BCL than another language. It also means that all the methods and properties that a class exposes are also available to all the languages. For instance, using the `System.Xml` namespace with Visual Basic .NET is the same as using the `System.Xml` namespace in C#. You only have to deal with the language syntax differences.

You can find information about all the namespaces in the .NET SDK documentation. There are quite a few, and you are not limited to these namespaces only. You can create your own namespaces to use within your applications. Third-party namespaces are also available on the market. 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 touches on a number of classes you can use as you build your ASP.NET applications.

With such a large number of classes at your disposal, you may find yourself searching high and low for a particular class. You can look for a class in several ways. The first way 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 on 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 2003\SDK\v1.1\Bin`. The left-hand pane of the WinCV tool shows your search results, and the right pane shows the type definition. The type definition is shown in a C#-like syntax (see Figure 1-5).



```
// from module 'c:\windows\assembly\gac\system.web\1.0.5000.0__b03f5f7f11d50a3a\system.web.dll'
public class System.Web.UI.WebControls.Xml :
    System.Web.UI.Control,
    System.ComponentModel.IComponent,
    IDisposable,
    System.Web.UI.IParserAccessor,
    System.Web.UI.IDataBindingsAccessor
{

    // Fields

    // Constructors
    public Xml();

    // Properties
    public Control BindingContainer { get; }
    public string ClientID { virtual get; }
    public ControlCollection Controls { virtual get; }
    public XmlDocument Document { get; set; }
    public string DocumentContent { get; set; }
    public string DocumentSource { get; set; }
    public bool EnableViewState { virtual get; virtual set; }
    public string ID { virtual get; virtual set; }
    public Control NamingContainer { virtual get; }
    public Page Page { virtual get; virtual set; }
    public Control Parent { virtual get; }
    public ISite Site { virtual get; virtual set; }
    public string TemplateSourceDirectory { virtual get; }
    public XslTransform Transform { get; set; }
    public XsltArgumentList TransformArgumentList { get; set; }
    public string TransformSource { get; set; }
    public string UniqueID { virtual get; }
    public bool Visible { virtual get; virtual set; }

    // Events
    public event EventHandler DataBinding;
    public event EventHandler Disposed;
    public event EventHandler Init;
    public event EventHandler Load;
    public event EventHandler PreRender;
    public event EventHandler Unload;

    // Methods
    public virtual void DataBind();
    public virtual void Dispose();
    public virtual bool Equals(object obj);
    public virtual System.Web.UI.Control FindControl(string id);
    public virtual int GetHashCode();
    public Type GetType();
    public virtual bool HasControls();
    public void RenderControl(System.Web.UI.HtmlTextWriter writer);
    public string ResolveUrl(string relativeUrl);
    public void SetRenderMethodDelegate(System.Web.UI.RenderMethod renderMethod);
    public virtual string ToString();
} // end of System.Web.UI.WebControls.Xml
```

Figure 1-5: 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 long list of classes in the left pane. Clicking the first choice, WebService, you are shown the type definition for the WebService class in the right 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.

What's New in .NET 1.1?

The .NET Framework 1.0 was introduced in 2002, and the .NET Framework 1.1 was introduced in 2003. As stated, the .NET Framework 1.1 is a minor release of the Framework. Everything that is discussed previously about the Framework can be true for either version.

Microsoft has tried hard to ensure that nothing breaks your code if you migrate your ASP.NET application from Version 1.0 to Version 1.1, but there are a few places in which you find breaking changes.

on the web

It's beyond the scope of this chapter to point out all the possible places that your code may break if you upgrade to .NET 1.1. You can find a complete list of changes and breaking changes on the GotDotNet Web site at `www.gotdotnet.com`**. Be sure to always create a staging server and completely test the upgrade before you move your application to a live environment with the upgrade.**

The following sections take a look at some of the major differences that you find in Version 1.1 of the .NET Framework as well as in the new Visual Studio .NET 2003.

Side-by-side as promised!

Microsoft touted from the beginning that it was solving the problem of DLL Hell with .NET because .NET could run multiple versions of your assemblies on the same server. The .NET Framework 1.1 is no different! You can now have multiple versions of the .NET Framework on your server.

This means that some of your ASP.NET applications can now run off the .NET Framework 1.0, while other applications might run off .NET Framework 1.1. This also means that you can have multiple versions of your application—one running off the .NET Framework 1.0 while another is running off the newer .NET Framework 1.1.

There are a couple of ways of getting the .NET Framework 1.1 to install on your server. The first, and one of the easiest, is to simply use Windows Server 2003. By default, this server contains both versions of the .NET Framework.

Another way of getting the .NET Framework 1.1 is to install Visual Studio .NET 2003. Installing this new IDE also installs the .NET Framework 1.1 for you. Then finally, another way of getting the new version of the .NET Framework is to go to Microsoft's site at `http://msdn.microsoft.com/netframework` and download it directly.

After it is installed, if you go to `C:\WINDOWS\Microsoft.NET\Framework`, you find both versions of .NET installed on your machine.

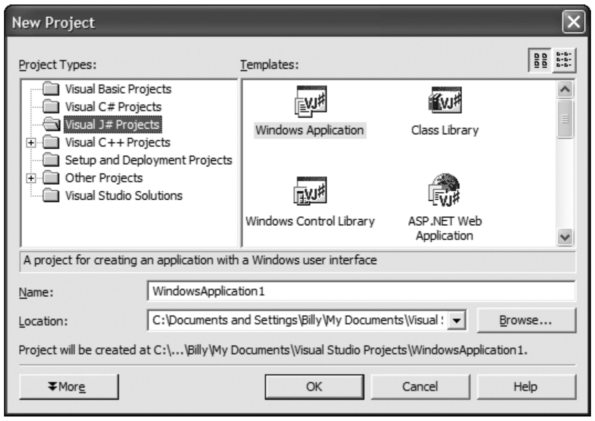
Another language: Visual J#

When you install Visual Studio .NET 2003, it now automatically includes Visual J#. This language used to be a separate download and install. Now that it is built in, there is hope that this language will have a larger adoption rate than it has had as a separate download.

Visual J#, or simply J# (pronounced *J-sharp*) is the next version of the Visual J++ language and is meant to be very much like the Java language—an ideal starting point in the

.NET world for Java developers. Although similar to Java, it has no support for the Java-runtime libraries. You might think of J# as Java, but with the .NET Framework class libraries used in place of the Java runtime libraries.

When you install Visual Studio .NET, you now have an entire J# section showing in the New Project dialog box (see Figure 1-6).



A project for creating an application with a Windows user interface

Name: WindowsApplication1

Location: C:\Documents and Settings\Billy\My Documents\Visual S

Project will be created at C:\...\Billy\My Documents\Visual Studio Projects\WindowsApplication1.

Figure 1-6: Selecting a J# project from the New Project dialog box.

Using J#, you can create classes, Windows Forms applications, ASP.NET Web applications, and XML Web services—just as you can with the other .NET languages. Because J# is part of the same family of languages, you can feel confident that you also have support for the core functions from the .NET Framework. You can also use J# in the same cross-language ways that you might use the other .NET-compliant languages—for example, using a J# class in your Visual Basic .NET application.

Just as the built-in Visual Basic .NET and C# compilers take applications that you build and convert them to IL, the built-in J# compiler also performs that same function.

You can see the compilers at `C:\WINDOWS\Microsoft.NET\Framework\v1.1.xxxx`. The Visual Basic .NET compiler is `vbc.exe`, the C# compiler is `csc.exe`, and the new J# compiler is `vjc.exe`.

Built-in mobility

You can now work with the mobile world in two ways, using the built-in features of the .NET Framework 1.1 and Visual Studio .NET 2003. The first is the now-native support for ASP.NET mobile controls.

Before the .NET Framework 1.1, to get mobile support in the ASP.NET you had to download the Microsoft Mobile Internet Toolkit (MMIT). Now you don't need a separate download because it is built right in.

You can see this when you create a new project. You now have the capability to create an ASP.NET mobile Web application (see Figure 1-7). Opening this type of project gives you a design surface and a large collection of mobile server controls.

Figure 1-7: The ASP.NET mobile Web application project type.

In addition to mobility in the Web world, you can also build mobile client applications—applications targeted for the Compact Framework. You can create this kind of project directly from the New Project dialog box by creating a smart device application. Figure 1-8 shows the icon that indicates this type of project.

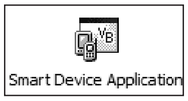


Figure 1-8: A smart device application project type.

You use this project type to create applications for the Pocket PC or any Windows CE device. These thick-client applications can utilize the Compact Framework, a trimmed-down version of the .NET Framework.

Two new data providers

An exciting set of capabilities in ADO.NET for Version 1.1 is the addition of two new data providers—one for ODBC and another for Oracle.

The ODBC data provider was an extra install for developers working on Version 1.0 of the .NET Framework, but it is now included in the default install of Version 1.1 of the Framework.

You can get at this new data provider by using the `System.Data.Odbc` namespace. If you are using the namespace as an extra install with Version 1.0 of the .NET Framework, the namespace is `Microsoft.Data.Odbc`. Using this data provider enables you to access any data source through the ODBC data provider on the server.

The second new data provider is the Oracle data provider. This data provider was also an extra download for users of Version 1.0 of the .NET Framework, but is now a default installation with Version 1.1 of the .NET Framework. To use the new Oracle data provider with your applications, you first refer to the `System.Data.OracleClient.dll` in your project.

These two data providers also give you ODBC and Oracle data connections, data adapters, data readers, and more.

Internet Protocol Version 6 support

Presently, much of the Internet runs using Internet Protocol Version 4, normally referred to as IPv4. This is what gives us IP addresses such as `255.255.255.255`. Internet Protocol Version 6, or IPv6, was created in 1995 to address many of the problems that the world was facing with IPv4—mostly due to the fact that we were running out of unique IP addresses under IPv4. Switching to IPv6 greatly expands the number of new IP address for the growing Internet world. Support for IPv6 is found in the `System.Net` namespace.

Changes to Visual Studio .NET

Visual Studio .NET 2003 has also changed along with the .NET Framework. One change is that when you install Visual Studio .NET 2003 on a machine with Visual Studio .NET 2002, the two IDEs install side by side. If you open a project that was originally built using VS.NET 2002 in VS.NET 2003, the project is automatically converted to a 2003 project.

caution **If you convert your Visual Studio .NET project from 2002 to 2003, you will be unable to convert it back to 2002.**

Visual Studio .NET 2003 has the same look and feel to it but with some new graphics and a new Start page. Some of the dialog boxes have changed a bit, but you meet some of the best changes when you start coding.

First and foremost, IntelliSense is greatly improved in Visual Studio .NET 2003. Now instead of always starting at the top of a long list of choices, IntelliSense uses a history of your choices to start with the most logical ones. This one new feature greatly speeds up your development time. In addition to the changes with IntelliSense, you also find new ways in which Visual Studio .NET completes code statements for you, speeding up your development process.

Changes to the Visual Basic .NET features

Using Visual Studio .NET 2003, you can now take chunks of Visual Basic 6.0 code, place the code in your project, and use the new Upgrade Visual Basic 6 Code dialog box to update the code to Visual Basic .NET for you.

Another new feature in the Visual Basic .NET world is the capability to upgrade older Visual Basic 6.0 WebClass projects to ASP.NET Web Application projects.

Summary

Whether you are brand new to the .NET world or have been with .NET since the beginning, you will find the ASP.NET Version 1.1 world an exciting and wonderful place in which to create your Web-based applications.

The .NET Framework 1.1 offers a powerful and revolutionary way to create your Web applications. After working in this new environment, you will never turn back.

The rest of this book takes you down some exciting paths with ASP.NET. Sit back, read, code, and enjoy!