1) The .NET Framework

As you've already learned, the .NET Framework is really a cluster of several technologies:

The .NET languages: These include Visual Basic, C#, F#, and C++, although third-party developers have created hundreds more.

The Common Language Runtime (CLR): This is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.

The .NET Framework class library: The class library collects thousands of pieces of prebuilt functionality that you can "snap in" to your applications. These features are sometimes organized into technology sets, such as ADO.NET (the technology for creating database applications) and Windows Presentation Foundation (WPF, the technology for creating desktop user interfaces).

ASP.NET: This is the engine that hosts the web applications you create with .NET, and supports almost any feature from the .NET Framework class library. ASP.NET also includes a set of web-specific services, such as secure authentication and data storage.

Visual Studio: This optional development tool contains a rich set of productivity and debugging features. Visual Studio includes the complete .NET Framework, so you won't need to download it separately.

Sometimes the division between these components isn't clear. For example, the term *ASP.NET* is sometimes used in a narrow sense to refer to the portion of the .NET class library used to design web pages. On the other hand, ASP.NET also refers to the whole topic of .NET web applications, which includes .NET languages and many fundamental pieces of the class library that aren't web-specific. (That's generally the way we use the term in this book. Our exhaustive examination of ASP.NET includes .NET basics, the C# language, and topics that any .NET developer could use, such as component-based programming and database access.)

Figure 1-4 shows the .NET class library and CLR—the two fundamental parts of .NET.

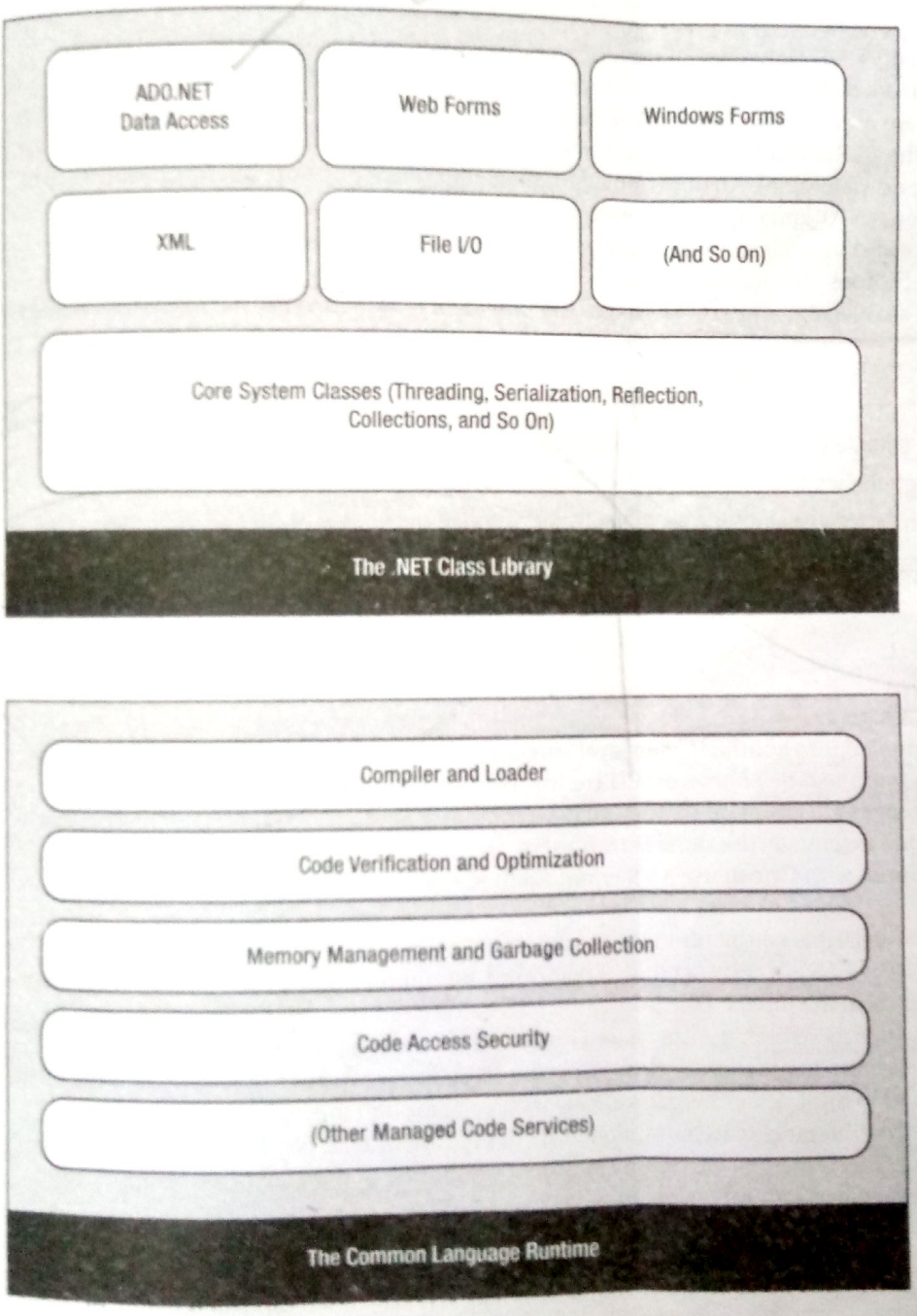

Figure 1-4. The .NET Framework

Intermediate Language

All the .NET languages are compiled into another lower-level language before the code is executed. This lower-level language is the *Common Intermediate Language* (CIL, or just IL). The CLR, the engine of .NET, uses only IL code. Because all .NET languages are based on IL, they all have profound similarities. This is the reason that the VB and C# languages provide essentially the same features and performance. In fact, the languages are so compatible that a web page written with C# can use a VB component in the same way it uses a C# component, and vice versa.

The .NET Framework formalizes this compatibility with something called the *Common Language Specification* (CLS). Essentially, the CLS is a contract that, if respected, guarantees that a component written in one .NET language can be used in all the others. One part of the CLS is the *common type system* (CTS), which defines the rules for data types such as strings, numbers, and arrays that are shared in all .NET languages. The CLS also defines object-oriented ingredients such as classes, methods, events, and quite a bit more. For the most part, .NET developers don't need to think about how the CLS works, even though they rely on it every day.

Figure 1-5 shows how the .NET languages are compiled to IL. Every EXE or DLL file that you build with a .NET language contains IL code. This is the file you deploy to other computers. In the case of a web application, you deploy your compiled code to a live web server.

The Common Language Runtime

The CLR is the engine that supports all the .NET languages. All .NET code runs inside the CLR. This is true whether you're running a Windows application or a web service. For example, when a client requests an ASP.NET web page, the ASP.NET service runs inside the CLR environment, executes your code, and creates a final HTML page to send to the client.

Not only does the CLR execute code, but it also provides a whole set of related services such as code verification, optimization, and object management. The implications of the CLR are wide-ranging:

Deep language integration: VB and C#, like all .NET languages, compile to IL. In other words, the CLR makes no distinction between different languages—in fact, it has no way of knowing what language was used to create an executable. This is far more than mere language compatibility; it's language *integration*.

Side-by-side execution: The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed, meaning that you're able to upgrade to new versions of ASP.NET without replacing the current version or needing to rewrite your applications.

Fewer errors: Whole categories of errors are impossible with the CLR. For example, the CLR prevents many memory mistakes that are possible with lower-level languages such as C++.

Along with these truly revolutionary benefits, the CLR has some potential drawbacks. Here are two issues that are sometimes raised by new developers but aren't always answered:

Performance: A typical ASP.NET application is extremely fast, because ASP.NET code is compiled to machine code before it's executed. However, processor-crunching algorithms still can't match the blinding speed of well-written C++ code, because the CLR imposes some additional overhead. Generally, this is a factor only in a few performance-critical high-workload applications (such as real-time games). With high-volume web applications, the potential bottlenecks are rarely processor-related but are usually tied to the speed of an external resource such as a database or the web server's file system. With ASP.NET caching and some well-written database code, you can ensure excellent performance for any web application.

Code transparency: IL is much easier to disassemble, meaning that if you distribute a compiled application or component, other programmers may have an easier time determining how your code works. This isn't much of an issue for ASP.NET applications, which aren't distributed but are hosted on a secure web server.

The .NET Class Library

The .NET class library is a giant repository of classes that provide prefabricated functionality for everything from reading an XML file to sending an e-mail message. If you've had any exposure to Java, you may already be familiar with the idea of a class library. However, the .NET class library is more ambitious and comprehensive than just about any other programming framework. Any .NET language can use the .NET class library's features by interacting with the right objects. This helps encourage consistency among different .NET languages and removes the need to install numerous components on your computer or web server.

Some parts of the class library include features you'll never need to use in web applications (such as the classes used to create desktop applications with Windows interfaces). Other parts of the class library are targeted directly at web development. Still more classes can be used in various programming scenarios and aren't specific to web or Windows development. These include the base set of classes that define common variable types and the classes for data access, to name just a few. You'll explore the .NET Framework throughout this book.

You can think of the class library as a well-stocked programmer's toolkit. Microsoft's philosophy is that it will provide the tedious infrastructure so that application developers need only to write business-specific code. For example, the .NET Framework deals with thorny issues such as database transactions and concurrency, making sure that hundreds or thousands of simultaneous users can request the same web page at once. You just add the logic needed for your specific application.

Visual Studio

The last part of .NET is the Visual Studio development tool, which provides a rich environment where you can rapidly create advanced applications. Although in theory you could create an ASP.NET application without Visual Studio (for example, by writing all the source code in a text editor and compiling it with .NET's command-line compilers), this task would be tedious, painful, and prone to error. For that reason, all professional ASP.NET developers use a design tool such as Visual Studio.

Some of the features of Visual Studio include the following:

Page design: You can create an attractive page with drag-and-drop ease by using Visual Studio's integrated web form designer. You don't need to understand HTML.

Automatic error detection: You could save hours of work when Visual Studio detects and reports an error before you run your application. Potential problems are underlined, just like the "spell-as-you-go" feature found in many word processors.

Debugging tools: Visual Studio retains its legendary debugging tools, which allow you to watch your code in action and track the contents of variables. And you can test web applications just as easily as any other application type, because Visual Studio has a built-in web server that works just for debugging.

IntelliSense: Visual Studio provides statement completion for recognized objects and automatically lists information such as function parameters in helpful tooltips.

You'll learn about all these features in Chapter 4, when you consider the latest version of Visual Studio. It's also important to note that Visual Studio is available in several editions:

Visual Studio Express for Web: This is a completely free version of Visual Studio that's surprising capable. Its main limitation is that it allows you to build web applications and components only, not other types of .NET programs (for example, Windows applications).

The HTML Control Classes

Before you can continue any further with the currency converter, you need to know about the control objects you've created. All the HTML server controls are defined in the System.Web.UI.HtmlControls namespace. Each kind of control has a separate class. Table 5-3 describes the basic HTML server controls and shows you the related HTML element.

Table 5-3. *The HTML Server Control Classes*

Class Name	HTML Element	Description
HtmlForm	<form>	Wraps all the controls on a web page. All ASP.NET server controls must be placed inside an HtmlForm control so that they can send their data to the server when the page is submitted. Visual Studio adds the <form> element to all new pages. When designing a web page, you need to ensure that every other control you add is placed inside the <form> section.
HtmlAnchor	<a>	A hyperlink that the user clicks to jump to another page.
HtmlImage	<img>	A link that points to an image, which will be inserted into the web page at the current location.
HtmlTable, HtmlTableRow, and HtmlTableCell	<table>, <tr>, <th>, and <td>	A table that displays multiple rows and columns of static text.
HtmlInputButton, HtmlInputSubmit, and HtmlInputReset	<input type="button">, <input type="submit">, and <input type="reset">	A button that the user clicks to perform an action (HtmlInputButton), submit the page (HtmlInputSubmit), or clear all the user-supplied values in all the controls (HtmlInputReset).
HtmlButton	<button>	A button that the user clicks to perform an action. This is not supported by all browsers, so HtmlInputButton is usually used instead. The key difference is that the HtmlButton is a container element. As a result, you can insert just about anything inside it, including text and pictures. The HtmlInputButton, on the other hand, is strictly text-only.
HtmlInputCheckBox	<input type="checkbox">	A check box that the user can select or clear. Doesn't include any text of its own.
HtmlInputRadioButton	<input type="radio">	A radio button that can be selected in a group. Doesn't include any text of its own.
HtmlInputText and HtmlInputPassword	<input type="text"> and <input type="password">	A single-line text box enabling the user to enter information. Can also be displayed as a password field (which displays bullets instead of characters to hide the user input).
HtmlTextArea	<textarea>	A large text box for typing multiple lines of text.
HtmlInputImage	<input type="image">	Similar to the <img> tag, but inserts a "clickable" image that submits the page. Using server-side code, you can determine exactly where the user clicked in the image—a technique you'll consider later in this chapter.
HtmlInputFile	<input type="file">	A Browse button and text box that can be used to upload a file to your web server, as described in Chapter 17.

(continued)

Table 5-3. (*continued*)

Class Name	HTML Element	Description
HtmlInputHidden	<input type="hidden">	Contains text information that will be sent to the server when the page is posted back but won't be visible in the browser.
HtmlSelect	<select>	A drop-down or regular list box enabling the user to select an item.
HtmlHead and HtmlTitle	<head> and <title>	Represents the header information for the page, which includes information about the page that isn't actually displayed in the page, such as search keywords and the web page title. These are the only HTML server controls that aren't placed in the <form> section.
HtmlGenericControl	Any other HTML element.	This control can represent a variety of HTML elements that don't have dedicated control classes. For example, if you add the `runat="server"` attribute to a element, it's provided to your code as an HtmlGenericControl object. You can identify the type of element by reading the TagName property, which stores a string (for example, "div").

3)b)

Table 5-2. ASP.NET Folders

Directory	Description
App_Browsers	Contains .browser files that ASP.NET uses to identify the browsers that are using your application and determine their capabilities. Usually, browser information is standardized across the entire web server, and you don't need to use this folder. For more information about ASP.NET's browser support—which is an advanced feature that most ordinary web developers can safely ignore—refer to *Pro ASP.NET 4.5 in C#* (Apress, 2012).
App_Code	Contains source code files that are dynamically compiled for use in your application.
App_GlobalResources	Stores global resources that are accessible to every page in the web application. This directory is used in localization scenarios, when you need to have a website in more than one language. Localization isn't covered in this book, and you can refer to *Pro ASP.NET 4.5 in C#* for more information.
App_LocalResources	Serves the same purpose as App_GlobalResources, except these resources are accessible to a specific page only.
App_WebReferences	Stores references to web services, which are remote code routines that a web application can call over a network or the Internet.
App_Data	Stores data, including SQL Server Express database files (as you'll see in Chapter 14). Of course, you're free to store data files in other directories.
App_Themes	Stores the themes that are used to standardize and reuse formatting in your web application. You'll learn about themes in Chapter 12.
Bin	Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses. For example, if you develop a custom component for accessing a database (see Chapter 22), you'll place the component here. ASP.NET will automatically detect the assembly, and any page in the web application will be able to use it.

The WebControl Base Class

Most web controls begin by inheriting from the WebControl base class. This class defines the essential functionality for tasks such as data binding and includes some basic properties that you can use with almost a web control, as described in Table 6-2.

3)a)

Table 6-2. *WebControl Properties*

Property	Description
AccessKey	Specifies the keyboard shortcut as one letter. For example, if you set this to Y, the Alt+Y keyboard combination will automatically change focus to this web control (assuming the browser supports this feature).
BackColor, ForeColor, and BorderColor	Sets the colors used for the background, foreground, and border of the control. In most controls, the foreground color sets the text color.
BorderWidth	Specifies the size of the control border.
BorderStyle	One of the values from the BorderStyle enumeration, including Dashed, Dotted, Double, Groove, Ridge, Inset, Outset, Solid, and None.
Controls	Provides a collection of all the controls contained inside the current control. Each object is provided as a generic System.Web.UI.Control object, so you will need to cast the reference to access control-specific properties.
Enabled	When set to false, the control will be visible, but it will not be able to receive user input or focus.
EnableViewState	Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag (in the .aspx page) every time the page is posted back. If this is set to true (the default), the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
Font	Specifies the font used to render any text in the control as a System.Web.UI.WebControls.FontInfo object.
Height and Width	Specifies the width and height of the control. For some controls, these properties will be ignored when used with older browsers.
ID	Specifies the name that you use to interact with the control in your code (and also serves as the basis for the ID that's used to name the top-level element in the rendered HTML).
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
TabIndex	A number that allows you to control the tab order. The control with a TabIndex of 0 has the focus when the page first loads. Pressing Tab moves the user to the control with the next lowest TabIndex, provided it is enabled. This property is supported only in Internet Explorer.
ToolTip	Displays a text message when the user hovers the mouse above the control. Many older browsers don't support this property.
Visible	When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

4)

PreInit - As the name suggests, this event happens just before page initialization event starts. IsPostBack, IsCallback and IsCrossPagePostBack properties are set at this stage. This event allows us to set the master page and theme of a web application dynamically. PreInit is extensively used when working with dynamic controls.

Init - Page Init, event occurs after the Init event, of all the individual controls on the webform. Use this event to read or initialize control properties. The server controls are loaded and initialized from the Web form's view state.

InitComplete - As the name says, this event gets raised immediately after page initialization.

PreLoad - Happens just before the Page Load event.

Load - Page Load event, occurs before the load event of all the individual controls on that webform.

Control Events - After the Page load event, the control events like button's click, dropdownlist's selected index changed events are raised.

Load Complete - This event is raised after the control events are handled.

PreRender - This event is raised just before the rendering stage of the page.

PreRenderComplete - Raised immediately after the PreRender event.

Unload - Raised for each control and then for the page. At this stage the page is, unloaded from memory.

Error - This event occurs only if there is an unhandled exception.

To see the events execution order, create a new asp.net web project, and copy the following code.

5. Web Request