Thanks for downloading this chapter from Murach’s ASP.NET 4.6 Web Programming with C# 2015. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its “how-to” headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our website. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!
What developers have said about previous editions

“This book enabled me to write custom websites for two customers, which is exactly what I wanted it to do. No previous book did such an excellent job of demonstrating how to handle security and database classes.”

Charles Profitt, Developer, New York

“I purchased several ASP.NET books but didn’t really learn it until I purchased this one.”

J.R., Enterprise Developer, Florida

“The author does a fantastic job of walking readers through the fundamentals, making the process feel like you’re sitting down with a skilled instructor who’s moving you from one example to the next.”

Muhammad Riaz, Baton Rouge Oracle User Group

“Another awesome book from Murach. Their format makes learning new material easier, and their code examples WORK.”

Posted at an online bookseller

“I especially appreciated the workaround insights for common hang-ups, like dealing with back-button refreshes. These gems alone will be worth the price of the book for many first-time .NET web authors.”

5-star review by Mike Riley, DevProConnections.com

“With this book, you will not have to rely on Google and the internet for your tutorials any longer.”

James Payne, www.aspfree.com

“Great book. I’m hard to please, and as a programmer and engineer, I was surprised by how informative this book was.”

Posted at an online bookseller
The essence of ASP.NET programming

This section presents the essential skills for designing, coding, and testing ASP.NET web applications. After chapter 1 introduces you to the concepts and terms that you need to know for ASP.NET programming, chapter 2 shows you how to develop a one-page web application with ASP.NET. That includes designing the form for the application and writing the C# code that makes it work, and that gets you off to a fast start.

Next, chapter 3 shows you the right way to use HTML5, CSS3, and Bootstrap with an ASP.NET application, and chapter 4 shows you how to develop a two-page Shopping Cart application that gets product data from a database. At that point, you’ll know how to build multi-page applications. Then, chapter 5 shows you how test and debug ASP.NET applications.

When you finish all five chapters, you’ll be able to develop real-world applications of your own. You’ll have a solid understanding of how ASP.NET works. You’ll be ready for rapid progress as you read any of the other sections of the book...and you can read those sections in whatever sequence you prefer.
An introduction to ASP.NET programming

This chapter introduces you to the basic concepts of web programming and ASP.NET. Here, you’ll learn how web applications work and what software you need for developing ASP.NET web applications. You’ll also see how the HTML code for a web form is coordinated with the C# code that makes the web form work the way you want it to. When you finish this chapter, you’ll have the background that you need for learning how to develop ASP.NET web applications with Visual Studio 2015.
An introduction to web applications

A web application consists of a set of web pages that are generated in response to user requests. The Internet has many different types of web applications, such as search engines, online stores, auctions, news sites, social sites, and games.

Two pages of a Shopping Cart application

Figure 1-1 shows two pages of an ASP.NET web application. In this case, the application is for an online store that lets users purchase Halloween products, including costumes, masks, and decorations. In chapter 4, you’ll learn how to build this application.

The first web page in this figure is used to display information about the products that are available from the Halloween store. To select a product, you use the drop-down list that’s below the banner at the top of the page. Then, the page displays information about the product including a photo, short and long descriptions, and the product’s price. The application gets the data for these pages from a database.

If you enter a quantity in the text box near the bottom of the page and click the Add to Cart button, the second page in this figure is displayed. This page lists the contents of your shopping cart and provides several buttons that let you remove items from the cart, clear the cart, return to the previous page to continue shopping, or proceed to a checkout page.

Of course, the complete Halloween Superstore application also contains other pages. For example, if you click the Check Out button in the second page, you’re taken to a page that lets you enter the information for completing the order. As you go through this book, you’ll learn how to add other pages to this application.
The Order page of a Shopping Cart application

The Cart page of a Shopping Cart application

Figure 1-1 Two pages of a Shopping Cart application
The components of a web application

The diagram in figure 1-2 shows that web applications consist of clients and a web server. The clients are the computers, tablets, and mobile devices that use the web applications. They access the web pages through programs known as web browsers. The web server holds the files that make up the pages of a web application.

A network is a system that allows clients and servers to communicate. The Internet is a large network that consists of many smaller networks. In a diagram like the one in this figure, the “cloud” represents the network or Internet that connects the clients and servers.

Networks can be categorized by size. A local area network (LAN) is a small network of computers that are near each other and can communicate with each other over short distances. Computers in a LAN are typically in the same building or adjacent buildings. This type of network is often called an intranet, and it can run web applications that are used throughout a company.

In contrast, a wide area network (WAN) consists of multiple LANs that have been connected. To pass information from one client to another, a router determines which network is closest to the destination and sends the information over that network. A WAN can be owned privately by one company or it can be shared by multiple companies.

An Internet service provider (ISP) is a company that owns a WAN that is connected to the Internet. An ISP leases access to its network to companies that need to be connected to the Internet. When you develop production web applications, you will often implement them through an ISP.

To access a web page from a browser, you can type a URL (Uniform Resource Locator) into the browser’s address area and press Enter. The URL starts with the protocol, which is usually HTTP. It is followed by the domain name and the folder or directory path to the file that is requested. If the file name is omitted in the URL, the web server looks for a default file in the specified directory. The default files usually include index.html, index.htm, default.html, and default.htm.
The components of a web application

- A web application consists of clients, a web server, and a network.
- The clients use programs known as web browsers to request web pages from the web server. Today, the clients can be computers, smart phones, or tablets.
- The web server returns the pages that are requested to the browser.
- A network connects the clients to the web server.
- To request a page from a web server, the user can type the address of a web page, called a URL, or Uniform Resource Locator, into the browser’s address area and then press the Enter key.
- A URL consists of the protocol (usually, HTTP), domain name, path, and file name. If you omit the protocol, HTTP is assumed. If you omit the file name, the web server will look for a file named index.html, index.htm, default.html, or default.htm.
- An intranet is a local area network (or LAN) that connects computers that are near each other, usually within the same building.
- The Internet is a network that consists of many wide area networks (WANs), and each of those consists of two or more LANs. Today, the Internet is often referred to as “the Cloud”, which implies that you don’t have to understand how it works.
- An Internet service provider (ISP) owns a WAN that is connected to the Internet.
How static web pages are processed

A static web page like the one in figure 1-3 is a web page that doesn’t change each time it is requested. This type of web page is sent directly from the web server to the web browser when the browser requests it. You can spot static pages in a web browser by looking at the extension in the address bar. If the extension is .htm or .html, the page is probably a static web page.

The diagram in this figure shows how a web server processes a request for a static web page. This process begins when a client requests a web page in a web browser. To do that, the user can either enter the URL of the page in the browser’s address bar or click a link in the current page that specifies the next page to load.

In either case, the web browser builds a request for the web page and sends it to the web server. This request, known as an HTTP request, is formatted using the HyperText Transfer Protocol (HTTP), which lets the web server know which file is being requested.

When the web server receives the HTTP request, it retrieves the requested file from the disk drive. This file contains the HTML (HyperText Markup Language) for the requested page. Then, the web server sends the HTML back to the browser as part of an HTTP response.

When the browser receives the HTTP response, it renders (translates) the HTML into a web page that is displayed in the browser. Then, the user can view the content. If the user requests another page, either by clicking a link or entering another URL into the browser’s address bar, the process begins again.
A static web page

How a web server processes a static web page

Description

- Hypertext Markup Language (HTML) is used to design the pages of a web application.
- A static web page is built from an HTML document that’s stored on the web server and doesn’t change. The file names for static web pages usually have .htm or .html extensions.
- When the user requests a static web page, the browser sends an HTTP request to the web server that includes the name of the file that’s being requested.
- When the web server receives the request, it retrieves the HTML for the web page and sends it back to the browser as part of an HTTP response.
- When the browser receives the HTTP response, it renders the HTML into a web page that is displayed in the browser.
How dynamic web pages are processed

A dynamic web page like the one in figure 1-4 is a page that’s created by a program on an application server. This program uses the data that’s sent with the HTTP request to generate the HTML that’s returned to the server. In this example, the HTTP request included the product code. Then, the program retrieved the data for that product from a database server, including the path to the photo for the product.

The diagram in this figure shows how a web server processes a dynamic web page. The process begins when the user requests a page in a web browser. To do that, the user can click a link that specifies the dynamic page to load or click a button that submits a form that contains the data that the dynamic page should process.

In either case, the web browser builds an HTTP request and sends it to the web server. This request includes whatever data the application needs for processing the request. If, for example, the user has entered data into a form, that data will be included in the HTTP request.

When the web server receives the HTTP request, the server examines the file extension of the requested web page to identify the application server that should process the request. The web server then forwards the request to that application server.

Next, the application server retrieves the appropriate program. It also loads any form data that the user submitted. Then, it executes the program. As the program executes, it generates the HTML for the web page. If necessary, the program will also request data from a database server and use that data as part of the web page it is generating.

When the program is finished, the application server sends the dynamically generated HTML back to the web server. Then, the web server sends the HTML back to the browser in an HTTP response.

When the web browser receives the HTTP response, it renders the HTML and displays the web page. Note, however, that the web browser has no way to tell whether the HTML in the HTTP response was for a static page or a dynamic page. It just renders the HTML.

When the page is displayed, the user can view the content. Then, when the user requests another page, the process begins again. The process that begins with the user requesting a web page and ends with the server sending a response back to the client is called a round trip.

When you build ASP.NET applications, Internet Information Services (IIS) is used for the web server, and ASP.NET is used for the application server. You’re also likely to use Microsoft’s SQL Server for the DBMS (database management system).
A dynamic web page

How a web server processes a dynamic web page

Description

- A dynamic web page is a web page that’s generated by a program running on a server.
- When a web server receives a request for a dynamic web page, it looks up the extension of the requested file and passes the request to the appropriate application server for processing.
- When the application server receives the request, it runs the appropriate program. Often, this program uses data that’s sent in the HTTP request to get related data from a database management system (DBMS) running on a database server.
- When the application server finishes processing the data, it generates the HTML for a web page and returns it to the web server. Then, the web server returns the HTML to the web browser as part of an HTTP response.
- The process that starts when a client requests a page and ends when the page is returned to the browser is called a round trip.
- When you build ASP.NET applications, Internet Information Services (IIS) is used for the web server and ASP.NET is used for the application server.

Figure 1-4 How dynamic web pages are processed
An introduction to ASP.NET development

In the topics that follow, you’ll be introduced to the main ASP.NET technologies and the differences between them, the three types of development environments you can work in, and more.

The two main ASP.NET technologies

Figure 1-5 summarizes the two main ASP.NET technologies. ASP.NET Web Forms were introduced in 2002 as a replacement for ASP (Active Server Pages), which is now called Classic ASP. In contrast to ASP, ASP.NET Web Forms let you work with a design model like the one for Windows Forms.

The primary focus of ASP.NET Web Forms is Rapid Application Development (RAD). It accomplishes this by letting web developers work with server controls on a design surface. Then, ASP.NET converts the server controls to HTML. In this book, you’ll learn to develop web applications with ASP.NET Web Forms.

In recent years, Microsoft added ASP.NET MVC to its web development offerings. It provides a way to implement the Model-View-Controller (MVC) pattern that offers separation of concerns and unit testing.

Separation of concerns refers to breaking an application into components so each one deals with a single concern. For example, one component can be responsible for communicating with the database, another for presenting information to users, and so on. Unit testing refers to code that tests whether other code does what it’s supposed to do.

Because the benefits of ASP.NET MVC are compelling, new ASP.NET web development is often done with MVC. That’s why the last chapter of this book presents an introduction to MVC. On the other hand, Web Forms development is easier than MVC development and may be more appropriate for smaller projects that need to be done quickly.

The two main types of Web Forms projects

In the web development world, the terms web site, web application, and web project are often used interchangeably. When you use Visual Studio 2015, though, these terms have specific meanings.

In ASP.NET Web Forms, a web project is either an ASP.NET Web Forms Site (or web site) or an ASP.NET Web Forms Application (or web application). The difference between the two is in how the projects are configured, compiled, and deployed, as indicated in figure 1-5.

Although there are pros and cons to each project type, Microsoft recommends that you use web application projects for new development, since they plan to focus their development efforts on that project type. Because of that, the projects for this book are all web application projects.
The two main ASP.NET technologies

- ASP.NET Web Forms
- ASP.NET MVC

The differences between the technologies

<table>
<thead>
<tr>
<th>Technology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP.NET Web Forms</td>
<td>A development environment similar to Windows Forms, with controls on a design surface. Its focus is on Rapid Application Development (RAD).</td>
</tr>
<tr>
<td>ASP.NET MVC</td>
<td>A development environment similar to PHP or classic ASP. It uses the Model-View-Controller (MVC) design pattern and the Razor templating engine for in-line data binding. Its focus is on separation of concerns and unit testing, and it gives the developer complete control over the HTML.</td>
</tr>
</tbody>
</table>

The two main types of ASP.NET Web Forms projects

- Web application projects
- Web site projects

The differences between the project types

<table>
<thead>
<tr>
<th>Project type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web application</td>
<td>A project file stores information about the application, and the application files are compiled into a single assembly. This assembly is then deployed to the web server. A few of the newest ASP.NET features are only available to web applications.</td>
</tr>
<tr>
<td>Web site</td>
<td>There is no project file. Rather, all the files that are in the root directory are included in the web site. The files are individually deployed to the web server, and the site is dynamically compiled the first time the web site is requested.</td>
</tr>
</tbody>
</table>

Description

- Microsoft has developed several ASP.NET technologies over the years. The two most popular are ASP.NET Web Forms and ASP.NET MVC.

- **Web Forms** is the oldest and most established technology. It provides for **RAD** (Rapid Application Development) by letting developers build web pages by working with controls on a design surface.

- **MVC** (Model-View-Controller) is relatively new to the .NET family. It addresses perceived weaknesses in Web Forms, such as inadequate **separation of concerns** and the difficulty of **unit testing**. Since ASP.NET MVC is becoming increasingly popular, the last chapter in this book provides an introduction to it.

- When you work with Web Forms, you can create either a **web application project** or a **web site project**. Microsoft recommends that new development be done using web application projects, and some new functionality is only available with this type of project.
Three environments for developing ASP.NET applications

Figure 1-6 shows three development environments for ASP.NET applications. In a standalone environment, a single computer serves as both the client and the server. In an intranet environment, the clients are connected to the server over an intranet. And in an Internet environment, the clients are connected to the server over the Internet.

In all three environments, the clients need an operating system like Windows 7, 8, or 10 that supports ASP.NET 4.6 development, the .NET Framework 4.5.2 or 4.6, and Visual Studio 2015. Since the .NET Framework comes with Windows 7, 8, and 10, and also with Visual Studio 2015, you don’t need to install it separately.

For the server, you need to install IIS as the application server and a database management system like SQL Server. In a standalone environment, you’re likely to use IIS Express and SQL Server Express LocalDB, which come with Visual Studio 2015. But in an intranet or Internet environment, you’re likely to use a full version of IIS and SQL Server.

In an intranet environment, the server also uses WebDAV (Web-based Distributed Authoring and Versioning). WebDAV provides the services that Visual Studio 2015 uses to communicate with the web application on the server. Normally, though, you don’t have to worry about this because the network manager sets this up.

In an Internet environment, the server also requires an FTP server, which is used to copy the files in a web application between the client computer and the server. The FTP server uses File Transfer Protocol (FTP) to perform the copy operations, and IIS can be configured to act as an FTP server as well as a web server. Here again, you usually don’t have to worry about this because the server manager sets this up.

The table in this figure shows that Visual Studio 2015 is available in three editions. Most professional developers will work with the Professional edition, but large development teams may use the Enterprise edition, which includes features that provide for specialized development roles such as architects, developers, and testers.

A free alternative is Visual Studio Community Edition. This edition is designed for individual developers, students, and hobbyists, and everything that you’ll learn in this book will work with the Community edition.

If you’re learning on your own, you will most likely work in a standalone environment using Visual Studio 2015 Community Edition, IIS Express, and SQL Server Express LocalDB, which are all free. If you’re working in a computer lab for a course, you will most likely work in an intranet environment, but it could also be an Internet environment. To install the client software that you’ll need for any of these environments, you can follow the procedures in appendix A.
**Standalone development**

- Windows 7 or later
- .NET Framework 4.6
- Visual Studio 2015
- IIS Express
- SQL Server Express LocalDB

**Intranet development**

- Windows 7 or later
- .NET Framework 4.6
- Visual Studio 2015
- LAN connection
- Windows Server 2008 R2 or later
- .NET Framework 4.6
- IIS 7.0 or later
- SQL Server
- WebDAV

**Internet development**

- Windows 7 or later
- .NET Framework 4.6
- Visual Studio 2015
- Internet connection
- Windows Server 2008 R2 or later
- .NET Framework 4.6
- IIS 7.0 or later
- FTP server
- SQL Server

### The three editions of Visual Studio 2015

<table>
<thead>
<tr>
<th>Edition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio Professional 2015</td>
<td>For individuals or small teams, it includes basic tools for testing, database deployment, and change and lifecycle management.</td>
</tr>
<tr>
<td>Visual Studio Enterprise 2015</td>
<td>For teams, it includes full testing, modeling, database, and lifecycle management tools.</td>
</tr>
</tbody>
</table>

### Description

- When you use standalone development, a single computer serves as client and server.

- When you use intranet development, a client communicates with a server over a local area network (LAN). For this, the server uses WebDAV (Web-based Distributed Authoring and Versioning).

- When you use Internet development, a client communicates with a server over the Internet. For this, the server requires an FTP server. The FTP server uses File Transfer Protocol (FTP) to transfer files between the client computer and the server.
The components of the .NET Framework

Because you should have a basic understanding of what the .NET Framework does as you develop applications, figure 1-7 summarizes its major components. As you can see, this framework is divided into two main components, the .NET Framework Class Library and the Common Language Runtime, and these components provide a common set of services for applications written in .NET languages like Visual Basic or C#.

The .NET Framework Class Library consists of classes that provide many of the functions that you need for developing .NET applications. For instance, the ASP.NET classes are used for developing ASP.NET web applications, and the Windows Forms classes are used for developing standard Windows applications. The other .NET classes let you work with databases, manage security, access files, and perform many other functions.

The Common Language Runtime, or CLR, provides the services that are needed for executing any application that’s developed with one of the .NET languages. This is possible because all of the .NET languages compile to a common Intermediate Language (or IL), which is stored in an assembly.

The CLR also provides the Common Type System that defines the data types that are used by all the .NET languages. That way, you can use the same data types no matter which .NET language you’re using to develop your applications.
The .NET Framework

.NET Applications
- Visual Basic
- Visual C#
- Visual C++
- Visual F#

.NET Framework

.NET Framework Class Library
- Windows Forms classes
- ASP.NET classes
- Other classes

Common Language Runtime
- Managed applications
- Common Type System
- Intermediate Language

Operating System and Hardware
- Windows 7
- Windows 8
- Windows 10
- Other Operating Systems

Description

- .NET applications work by using services of the .NET Framework. The .NET Framework, in turn, accesses the operating system and computer hardware.
- The .NET Framework consists of two main components: the .NET Framework Class Library and the Common Language Runtime.
- The .NET Framework Class Library provides pre-written code in the form of classes that are available to all of the .NET programming languages.
- The Common Language Runtime, or CLR, manages the execution of .NET programs by coordinating essential functions such as memory management and security.
- The Common Type System is a component of the CLR that ensures that all .NET applications use the same data types regardless of what programming languages are used.
- All .NET programs are compiled into Microsoft Intermediate Language (MSIL) or just Intermediate Language (IL), which is stored in an assembly. This assembly is then run by the CLR.

Figure 1-7 The components of the .NET Framework
Although it hasn’t been mentioned yet, a web application ends after it generates a web page. That means that any data maintained by the application, such as variables or control properties, is lost. In other words, HTTP doesn’t maintain the state of the application. This is illustrated in figure 1-8.

Here, you can see that a browser on a client requests a page from a web server. After the server processes the request and returns the page to the browser, it drops the connection. Then, if the browser makes additional requests, the server has no way to associate the browser with its previous requests. Because of that, HTTP is known as a stateless protocol.

Although HTTP doesn’t maintain state, ASP.NET provides several ways to do that, as summarized in this figure. First, you can use view state to maintain the values of server control properties. For example, you can use view state to preserve the values of the items in a drop-down list. Because ASP.NET implements view state by default, you don’t need to write any special code to use it.

Second, you can use session state to maintain data between executions of an application. To make this work, ASP.NET creates a session state object that is kept on the server whenever a user starts a new session. This session object contains a unique session ID, and this ID is sent back and forth between the server and the browser each time the user requests a page. Then, when the server receives a new request from a user, it can retrieve the right session object for that user. In the code for your web forms, you can add data items to the session object so their previous values are available each time a web form is executed.

Third, you can use an application state object to save application state data, which applies to all of the users of an application. For example, you can use application state to maintain global counters or to maintain a list of the users who are currently logged on to an application.

Fourth, you can use server-side caching to save data. This is similar to application state in that the data saved in the cache applies to all users of an application. However, caching is more flexible than application state because you have control over how long the data is retained.

Last, you can use the individual user accounts of ASP.NET to keep track of user data. Although this is similar to session state, the data persists between user sessions because it is stored in a database.
Why state is difficult to track in a web application

Concepts

- **State** refers to the current status of the properties, variables, and other data maintained by an application for a single user. The application must maintain a separate state for each user currently accessing the application.

- HTTP is a stateless protocol. That means that it doesn’t keep track of state between round trips. Once a browser makes a request and receives a response, the application terminates and its state is lost.

Five ASP.NET features for maintaining state

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View state</td>
<td>Implemented by default, so no special coding is required. See chapter 2.</td>
</tr>
<tr>
<td>Session state</td>
<td>Uses a session state object that is created when a user starts a new session. The values in this object are available until the session ends. See chapter 4.</td>
</tr>
<tr>
<td>Application state</td>
<td>Uses an application state object that is created when an application starts. The values of this object are available to all users of the application until the application ends. See chapter 8.</td>
</tr>
<tr>
<td>Server-side caching</td>
<td>Like application state, the values in a server-side cache can be shared across an application. Unlike application state, a cache item is maintained only until its expiration time is reached. See chapter 8.</td>
</tr>
<tr>
<td>Individual user accounts</td>
<td>One account can be maintained for each user of an application. The data in an account is stored in a database and maintained from one user session to another. See chapter 20.</td>
</tr>
</tbody>
</table>

Description

- ASP.NET provides five ways to deal with the stateless protocol of a web application. The two that you’ll use the most are view state and session state.
How an ASP.NET application works

With that as background, you’re ready to learn more about how an ASP.NET application works. That’s why this topic presents a one-page Future Value application.

The user interface for the Future Value application

Figure 1-9 presents the user interface for a one-page application called the Future Value application. In ASP.NET, pages like this are called web forms. To make these pages work, each form contains ASP.NET server controls that let the user interact with the page. For instance, this page contains these server controls: a drop-down list, two text boxes, a label that displays the future value, and two buttons. It also uses validation controls that check the user entries for validity.

To use the Future Value application, the user selects a monthly investment amount from the drop-down list, enters data into the two text boxes, and clicks the Calculate button. Then, if the data is valid, the future value is displayed. Otherwise, error messages are displayed below the buttons. To clear the controls, the user can click the Clear button.

The processing for the Calculate and Clear buttons is done on the server. That means that when either button is clicked, the form is submitted to the server, the C# code on the server processes the data in the form, and the form is returned to the browser. In other words, clicking either button leads to a round trip.

For example, when the Calculate button is clicked, the C# code on the server calculates the future value, and the form is returned to the browser with the future value displayed. That’s a round trip. Similarly, when the Clear button is clicked, the C# code on the server resets the value in the drop-down list to 50 and clears the text boxes and the Future Value label. Then, the form is returned to the browser. That’s also a round trip.

In some cases, though, the form isn’t submitted to the browser when the Calculate button is clicked. That happens when JavaScript is enabled in the user’s browser and one or more entries are invalid. Then, the JavaScript code that has been generated from the validation controls runs in the browser, detects the invalid entries, and displays appropriate error messages...without submitting the form to the server. That saves a round trip.

What if JavaScript isn’t enabled in the user’s browser? Then, the form is submitted to the server, and the server checks the entries for validity. That too is done by the code that’s generated by the validation controls. In this example, that means a round trip occurs each time the user clicks the Calculate button if JavaScript is disabled. Fortunately, though, most browsers have JavaScript enabled.
The Future Value application after the user clicks the Calculate button

![Image of the Future Value application after the user clicks the Calculate button]

The Future Value application with error messages displayed

![Image of the Future Value application with error messages displayed]

Description

- To calculate the Future Value of a monthly investment, the user selects a value in the drop-down list, enters values into the two text boxes, and clicks the Calculate button.
- If JavaScript is enabled in the browser, it is used to check the user’s entries. If the entries are valid, the form is submitted to the server, the future value is calculated, and the page is returned to the browser with the future value displayed. If the entries are invalid, error messages are displayed by the JavaScript and the form isn’t submitted. If JavaScript isn’t enabled, the form is submitted to the server and the validation is done there.
- If the user clicks on the Clear button, the form is submitted to the server, the drop-down list is reset to 50, the text boxes are cleared, and the page is returned to the browser.
The files used by the Future Value application

Figure 1-10 presents the Future Value form as it appears in the Web Forms Designer that you use when you develop web forms with Visual Studio 2015. In chapter 2, you’ll learn how to use this Designer, but for now just try to get the big picture of how this form works.

If you look closely at the Designer window in the middle of Visual Studio, you can see the table that’s used for this form. You can also see the server controls: the drop-down list that’s used to select a monthly investment amount, text boxes for interest rate and number of years, a label for displaying the result of the future value calculation, and Calculate and Clear buttons.

If you look at the Solution Explorer to the right of the Designer window, you can see the folders and files that this application requires. These are summarized in the table in this figure.

The first three files in the table are for the web form. The file with aspx as the extension (Default.aspx) contains the code that represents the design of the form. This code consists of standard HTML code plus asp tags that define the server controls. This is called aspx code, because the file that contains the code has the aspx extension.

The file with aspx.cs as the extension (Default.aspx.cs) contains the C# code that controls the operation of the form. The cs file extension indicates that it is a C# file. This is called a code-behind file because it provides the code behind the web form.

The file with aspx.designer.cs as the extension (Default.aspx.designer.cs) contains the C# code that’s generated by Visual Studio when you add server controls to the form. This is called a designer file because it provides code that Visual Studio uses to work with server controls in the Web Forms Designer. You should never need to make changes to this file, but it can be interesting to see what’s in it.

The fourth and fifth files in the table in this figure are the configuration files. The Web.config file contains configuration information like which version of the .NET Framework is being used. The packages.config file contains information about the NuGet packages the application is using. You’ll learn about NuGet packages in chapter 2.

The sixth file is the jpg file for the logo that’s displayed at the top of the form. This file is in the Images folder that’s shown in the Solution Explorer.

These files and the Images folder are all that this one-page application requires. However, two other folders are often used for Web Forms applications. A Models folder is used for user classes, and an App_Data folder is used for databases or data files. You’ll see these used in the Shopping Cart application in chapter 4.
The Future Value form in Design view of Visual Studio 2015

Some of the files in the Future Value application

<table>
<thead>
<tr>
<th>Folder</th>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>Default.aspx</td>
<td>The aspx file for the default page.</td>
</tr>
<tr>
<td>(root)</td>
<td>Default.aspx.cs</td>
<td>The code-behind file for the default page.</td>
</tr>
<tr>
<td>(root)</td>
<td>Default.aspx.designer.cs</td>
<td>The generated code for the default page.</td>
</tr>
<tr>
<td>(root)</td>
<td>Web.config, packages.config</td>
<td>XML files that contain configuration data.</td>
</tr>
<tr>
<td>Images</td>
<td>MurachLogo.jpg</td>
<td>The logo image for the form.</td>
</tr>
</tbody>
</table>

Description

- For each web form in a web application, ASP.NET 4.6 keeps three files. The file with the aspx extension holds the HTML code and the asp tags for the server controls. The file with the aspx.cs extension is the code-behind file that contains the C# code for the form. And the file with the aspx.designer.cs extension contains code generated by Visual Studio when you add server controls to the aspx file.

- Every ASP.NET application also includes a Web.config file with configuration data, and most applications include a packages.config file that contains information about NuGet packages used by the application.

- The Future Value application also contains a folder named Images that contains the jpg file for the logo that’s displayed at the top of the page.

- Two other folders that you’ll find in many ASP.NET web applications are a Models folder for user classes, and an App_Data folder for databases or data files.
The aspx code for the Default form

To give you some idea of how aspx code works, figure 1-11 shows the aspx code for the Default form. Most of this code is generated by Visual Studio as you use the Web Forms Designer to design a form, so you don’t have to code it all yourself. But you should understand how this code works.

The first set of tags for each web form defines a page directive that provides four attributes. The Language attribute says that the language is C#. The AutoEventWireup attribute says that the event handlers will be called automatically when the events occur for a page. The CodeBehind attribute says that the code-behind file is named Default.aspx.cs. And the Inherits attribute specifies the class named Default in the Ch01FutureValue namespace.

The second set of tags defines a DOCTYPE declaration, which tells the browser that HTML5 will be used for this page. If you aren’t already using HTML5, you can learn more about it in chapter 3.

The html tags mark the beginning and end of the HTML document, and the head tags define the head section for the document. Here, the title tags define the title that is displayed in the title bar or tab of the browser when the page is run. In addition, the style tags, which aren’t shown in this example, define the styles used by the page.

The content of the web page itself is defined within the div tags, which are within the body and form tags. Notice that the first form tag includes a Runat attribute that’s assigned a value of “server.” That indicates that the form will be processed on the server by ASP.NET. This attribute is required for all ASP.NET web forms and all ASP.NET server controls.

The asp tags within the div tags define the server controls that appear on the page. Since these controls include the Runat attribute with a value of “server,” they will be processed on the server by ASP.NET. The last phase of this processing is generating the HTML for the controls so the page can be displayed by a browser.

The aspx code for the Calculate and Clear buttons includes an OnClick attribute. This attribute names the event handler that’s executed when the user clicks the button and the form is posted back to the server.

The aspx code for the Clear button includes a CausesValidation attribute. This attribute tells the page whether to fire the validation event used by the validation controls. Setting this attribute to False for the Clear button means that the data validation controls will do their work when you click Calculate, but not when you click Clear.

Within the form, an HTML table is used to format the server controls. Here, the first four rows include the server controls that accept the user entries and display the future value. The fifth row provides vertical spacing. And the last row includes the Calculate and Clear button controls.

After the table are four field validator controls that aren’t shown. These are server controls that provide validity checking, both in the browser with JavaScript and on the server with C#, and they cause error messages to be displayed when an error occurs. In the next chapter, you’ll learn how to build this web form with its controls.
The aspx file for the Default form (Default.aspx)

```csharp
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="Ch01FutureValue.Default" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
<title>Chapter 1: Future Value</title>
<style type="text/css">!!-- CSS code for the generated styles --></style>
</head>
<body>
<form id="form1" runat="server">
<div>
<img alt="Murach" class="style1" src="Images/MurachLogo.jpg" /><br />
<h1>401K Future Value Calculator</h1>
<table class="style2">
<tr>
<td class="style3">Monthly investment</td>
<td><asp:DropDownList ID="ddlMonthlyInvestment" runat="server" Width="106px"></asp:DropDownList></td>
</tr>
<tr>
<td class="style3">Annual interest rate</td>
<td><asp:TextBox ID="txtInterestRate" runat="server" Width="100px">3.0</asp:TextBox></td>
</tr>
<tr>
<td class="style3">Number of years</td>
<td><asp:TextBox ID="txtYears" runat="server" Width="100px">10</asp:TextBox></td>
</tr>
<tr>
<td class="style3">Future value</td>
<td><asp:Label ID="lblFutureValue" runat="server" Font-Bold="True"></asp:Label></td>
</tr>
<tr>
<td class="style3">&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td class="style3"><asp:Button ID="btnCalculate" runat="server" Text="Calculate" Width="100px" OnClick="btnCalculate_Click" /></td>
<td><asp:Button ID="btnClear" runat="server" Text="Clear" Width="100px" OnClick="btnClear_Click" CausesValidation="False" /></td>
</tr>
</table>
<br />
<!-- aspx code for the field validators -->
</div>
</form>
</body>
</html>

Figure 1-11 The aspx code for the Default form of the Future Value application
To give you some idea of how the C# code for a form works, figure 1-12 presents the code-behind file for the Default form. Here, I’ve highlighted the most important code.

The first two highlighted lines are generated by Visual Studio. The first line is a namespace declaration and has the same name as the project. The second line is a class declaration and has the same name as the form.

Because the class declaration uses the partial keyword, this is a partial class that must be combined with another partial class when it’s compiled. In fact, the code in this partial C# class is combined with the compiled code in its aspx file and designer file. The rest of this class declaration indicates that this class inherits the System.Web.UI.Page class, which is the .NET class that provides the basic functionality of ASP.NET pages.

Each time this web form is requested, ASP.NET initializes it and raises the Load event, which is handled by the Page_Load method. You will often see a page property called IsPostBack used in the Page_Load method to determine whether or not a page is being posted back. If the value is false, the page is being loaded for the first time.

For this application, if the page is being loaded for the first time, the code executes a loop that puts a range of dollar amounts into the drop-down list for monthly investment. Otherwise, nothing is done by this method.

Another page property that you will often use is called IsValid. This property indicates whether the page’s validation controls detect invalid data in the server controls when the Calculate button is clicked. This property is used in the btnCalculate_Click method that is executed when the user clicks on the Calculate button, which starts a postback.

If the IsValid property indicates that the data is valid, this method retrieves the values from the server controls, converts them to the proper data types, and sends them to the CalculateFutureValue method for processing. When that method returns the future value, the btnCalculate_Click method formats the result as currency and puts it in the future value label. Then, the form is returned to the browser.

The btnClear_Click method is executed when the user clicks on the Clear button. This too starts a postback. Then, after the Page_Load method is executed, the btnClear_Click method clears the server controls by setting the index of the drop-down list to 0 and setting the text box and label properties to empty strings.

Since this book assumes that you already know how to use C#, you should be able to follow the C# code in this figure. The new points to note are (1) the Page_Load method is executed each time the page is requested, (2) the IsPostBack property tells whether a page is being requested for the first time, and (3) the IsValid property tells whether the validator controls have detected invalid data.
The code-behind file for the Default form (Default.aspx.cs)

```csharp
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
// the rest of the default using directives

namespace Ch01FutureValue
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                for (int i = 50; i <= 500; i += 50)
                {
                    ddlMonthlyInvestment.Items.Add(i.ToString());
                }
            }
        }

        protected void btnCalculate_Click(object sender, EventArgs e)
        {
            if (IsValid) {
                int monthlyInvestment = Convert.ToInt32(ddlMonthlyInvestment.SelectedValue);
                decimal yearlyInterestRate = Convert.ToDecimal(txtInterestRate.Text);
                int years = Convert.ToInt32(txtYears.Text);
                decimal futureValue = this.CalculateFutureValue(monthlyInvestment, yearlyInterestRate, years);
                lblFutureValue.Text = futureValue.ToString("c");
            }
        }

        protected decimal CalculateFutureValue(int monthlyInvestment, decimal yearlyInterestRate, int years)
        {
            int months = years * 12;
            decimal monthlyInterestRate = yearlyInterestRate / 12 / 100;
            decimal futureValue = 0;
            for (int i = 0; i < months; i++)
            {
                futureValue = (futureValue + monthlyInvestment) * (1 + monthlyInterestRate);
            }
            return futureValue;
        }

        protected void btnClear_Click(object sender, EventArgs e)
        {
            ddlMonthlyInvestment.SelectedIndex = 0;
            txtInterestRate.Text = "";
            txtYears.Text = "";
            lblFutureValue.Text = "";
        }
    } // end class
} // end namespace
```

Figure 1-12 The C# code for the Default form of the Future Value application
Perspective

Now that you’ve read this chapter, you should have a general understanding of how ASP.NET applications work and what software you need for developing these applications. With that as background, you’re ready to learn how to develop ASP.NET applications of your own. You’ll start that process in the next chapter.

Terms

- web application
- web page
- client
- web browser
- web server
- network
- intranet
- LAN (local area network)
- Internet
- WAN (wide area network)
- ISP (Internet service provider)
- URL (Uniform Resource Locator)
- protocol
- domain name
- path
- static web page
- HTML (Hypertext Markup Language)
- HTTP request
- HTTP (HyperText Transfer Protocol)
- HTTP response
- render HTML in the browser
- dynamic web page
- application server
- database server
- DBMS (database management system)
- round trip
- IIS (Internet Information Services)
- Web Forms
- ASP (Active Server Pages)
- classic ASP
- RAD (Rapid Application Development)
- MVC (Model-View-Controller)
- separation of concerns
- unit testing
- web application project
- web site project
- WebDAV (Web-based Distributed Authoring and Viewing)
- FTP server
- FTP (File Transfer Protocol)
- .NET Framework
- .NET Framework Class Library
- CLR (Common Language Runtime)
- IL (Intermediate Language)
- compile
- assembly
- state
- stateless protocol
- view state
- session state
- web form
- server control
- .aspx code
- code-behind file
- designer file
- page directive
Summary

- A web application consists of a set of web pages that are run by clients, a web server, and a network. Clients use web browsers to request web pages from the web server. The web server returns the requested pages.
- A local area network (LAN), or intranet, connects computers that are near each other. By contrast, the Internet consists of many wide area networks (WANs).
- One way to access a web page is to type a URL (Uniform Resource Locator) into the address area of a browser and press Enter. A URL consists of the protocol (usually, HTTP), domain name, path, and file name.
- To request a web page, the web browser sends an HTTP request to the web server. Then, the web server gets the HTML for the requested page and sends it back to the browser in an HTTP response. Last, the browser renders the HTML into a web page.
- A static web page is a page that is the same each time it’s retrieved. In contrast, the HTML for a dynamic web page is generated by a server-side program, so its HTML can change from one request to another. Either way, HTML is returned to the browser.
- For ASP.NET applications, the web server is usually Internet Information Services (IIS) and ASP.NET is the application server. The web server may also use a database management system (DBMS) like SQL Server.
- One way to develop ASP.NET applications is to use Web Forms. This is similar to using Windows Forms and encourages Rapid Application Development (RAD).
- Another way to develop ASP.NET applications is to use ASP.NET MVC (Model-View-Controller). It provides better separation of concerns and unit testing.
- To develop ASP.NET applications on your own computer, you need Windows 7 or later, Microsoft .NET Framework 4.5.2 or 4.6, Visual Studio 2015, IIS Express, a DBMS like SQL Server Express LocalDB, and one or more browsers.
- The .NET Framework provides the services that ASP.NET applications use to access the operating system and computer hardware. Its main components are the Class Library and the Common Language Runtime (CLR).
- HTTP is called a stateless protocol because it doesn’t keep track of the data (state) between round trips. However, ASP.NET provides five ways to keep track of state including view state and session state.
- The pages in an ASP.NET application are called web forms. They contain server controls like drop-down lists, text boxes, labels, and buttons.
- Each page in an ASP.NET application consists of an aspx file for the HTML and server controls, an aspx.cs file for the C# in the code-behind file, and an aspx.designer.cs file for the generated C# code in the designer file.
- Before a web form can be run, its aspx and C# files are compiled into an assembly that consists of Intermediate Language (IL) that is run by the CLR.
Before you do the exercises for this book...

Before you do the exercises for this book, you should install the software that’s required for this book as well as the downloadable applications for this book. Appendix A shows how to do that.

Exercise 1-1  Use Visual Studio to run the Future Value application

In this exercise, you’ll run the Future Value application. This will test whether you’ve successfully installed the software and applications for this book.

Start Visual Studio and open the Future Value application

2. Use the File→Open→Project/Solution command. In the dialog box that’s displayed, navigate to this folder:
   C:\aspnet46_cs\Ex01FutureValue
   Then, select the Ch01FutureValue.sln file and click the Open button.

Run the Future Value application

3. Press F5 to run the application. That should display the Future Value form in Visual Studio’s default web browser.
4. Without changing the values that are displayed, click the Calculate button. This starts a postback that returns the page with the result of the calculation.
5. Click the Clear button to clear the values from the text box controls.
6. Click the Calculate button again. Then, note the error messages that are displayed. These messages were generated by the validation controls.
7. Click the Clear button again. Note that the error messages go away. That’s because the Clear button has its CausesValidation attribute set to False.
8. Select an investment amount from the drop-down list, enter an annual interest rate greater than 20 and a number of years greater than 45. Then, click the Calculate button to see the error messages that are displayed.
9. Change the interest rate to 5 and the number of years to 30. Then, Click the Calculate button to see that the future value is displayed, which means the entries were valid.
10. Experiment on your own if you like. When you’re through, close the browser and then use the File→Close Solution command to close the web application. Then, close Visual Studio.
How to build your .NET web programming skills

The easiest way is to let Murach’s ASP.NET 4.6 Web Programming with C# 2015 be your guide! So if you’ve enjoyed this chapter, I hope you’ll get your own copy of the book today. You can use it to:

- Teach yourself how to develop professional, bulletproof ASP.NET Web Forms applications that manage state, validate data, handle everyday business needs, do database processing, and are inviting and easy-to-navigate for users

- Pick up new skills whenever you want to or need to by focusing on material that's new to you

- Look up coding details or refresh your memory on forgotten details when you’re in the middle of developing a Web Forms application

- Loan to your colleagues who are always asking you questions about ASP.NET Web Forms programming

To get your copy, you can order online at www.murach.com or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you’ve ever tried, or send it back within 60 days for a full refund…no questions asked.

Thanks for your interest in Murach books!