1

Getting Started with ASP.NET 6.0

WHAT YOU WILL LEARN IN THIS CHAPTER:

- A brief history of ASP.NET and why it supports both Web Forms and MVC
- About the two frameworks, Web Forms and MVC
- How to install and use Visual Studio 2015
- The sample application that will be used throughout this book

CODE DOWNLOADS FOR THIS CHAPTER:

The wrox.com code downloads for this chapter are found at `www.wrox.com/go/beginningaspnetforvisualstudio` on the Download Code tab. The code is in the chapter 01 download and individually named according to the names throughout the chapter.

The Internet has become a critical part of life to millions of people across the world. This growth in the use of the Internet has been accelerating since the 1990s and will continue as technology and access becomes more affordable. The Internet has become the go-to source for shopping, leisure, learning, and communications. It has helped to both build new businesses and give revolutionaries the capability to spread their message to the rest of the world.

This growth means that there will be a long-term demand for people with the skills to build and maintain the next generation of web applications. As an increasing percentage of the world's business is accomplished with web applications, learning how to work on these applications is an obvious career move.

AN INTRODUCTION TO ASP.NET vNEXT

The Internet started off as a set of sealed, private networks designed to share information between research institutions across the United States. The primary users of this system were the research scientists in those labs. However, as the usefulness and flexibility of this information-sharing approach became obvious, interest grew exponentially. More and more institutions became involved, resulting in the evolution of standards and protocols to support the sharing of additional types of information. The initial networks quickly expanded as commercial entities became involved. Soon, Internet service providers were available, enabling regular, everyday people to access and share the burgeoning content of the Internet.

In the early days of the Internet, most content was created and stored statically. Each HTTP request would be for a specific page or piece of stored content, and the response would simply provide that content. Early application frameworks changed that model, enabling the dynamic generation of content based on a certain set of criteria sent as part of that request. This enabled content to be built from databases and other sources, exponentially increasing the usefulness of the Web. It was at this point that the general public, rather than only scientists, really started to take advantage of the Internet's enhanced usability.

ASP.NET is one of those early web application frameworks, with the first version of the .NET Framework released in 2002. The ASP part of the name stands for "Active Server Pages," which was Microsoft's initial web application framework that used server-side processing to create browser-readable HTML pages. The original ASP, now called "Classic ASP," allowed the developer to use VBScript to add scripting code to HTML. However, the code and the HTML were all intermingled together in a single file.

ASP.NET was considered a major enhancement at the time because it allowed for a much cleaner separation of the code-behind, the code that handles the processing and markup, the code handling the building of the display, than any of the other frameworks available at that time. There have been improvements to this initial ASP.NET framework with every new release of the .NET Framework.

In 2008 Microsoft introduced a new framework that supported a different approach to content creation and navigation: ASP.NET MVC. MVC stands for Model View Controller, and references a software design pattern that provides a more complete separation between the user interface and the processing code. The original framework became known as Web Forms. Even as the Internet content-creation technologies evolve, the way that the Internet runs stays surprisingly unchanged. The movement of the information from the server to the client follows a very simple protocol that has barely changed since the beginning of the Internet.

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is the application protocol that acts as the foundation for communications within the Internet. It defines the interaction between the client machine and the server as following a request-response model whereby the client machine requests, or asks for, a specific resource and the server responds with, or sends a reply about, the information as appropriate.

This request can be very simple, from "show me this picture," to something very complex, such as a transfer between your bank accounts. Figure 1-1 shows the outcome of that request—whether it is

displaying the picture for the first, simple request or whether it is displaying the receipt for the bank transfer from the second, more complex request.

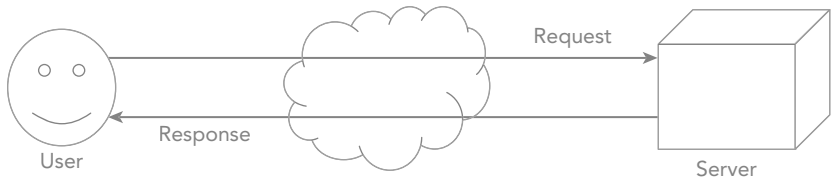

FIGURE 1-1: Request response

The HTTP protocol also defines what the requests and responses should look like. It includes methods, also known as verbs, which describe what kind of action should be taken on the item being requested. These verbs are not really used that much in ASP.NET Web Forms, but they are especially important in ASP.NET MVC because MVC uses these methods to identify the actions being taken on the requested object. The major verbs are listed in Table 1-1.

TABLE 1-1: Most Frequently Used HTTP Verbs

NAME	DESCRIPTION
GET	A GET is a request for a resource. It should retrieve that resource without any other effect resulting from taking that action. You should be able to GET a resource multiple times.
POST	A POST indicates that there is information included in the request that should create a new version of the resource. By definition, any item posted should create a new version, so passing in the same information multiple times should result in multiple instances of that object being created.
PUT	A PUT indicates that the information included in the request should change already existing items. The definition also allows the server to create a new item if the item that is expected to be changed has not already been created. This is different from the POST verb because a new item is only created when the request includes new information.
DELETE	The DELETE verb indicates that the specified resource should be deleted. This means that upon deletion, a GET or PUT to that resource will fail.

An HTTP request includes the following:

- A request line. For example, `GET/images/RockMyWroxLogo.png HTTP/1.1` requests a resource called /images/RockMyWroxLogo.png from the server.
- Request header fields, such as `Accept-Language: en`

- An empty line
- An optional message body; when using either the POST or PUT verbs, the information needed to create the object is generally put in this message body

An HTTP response includes these items:

- A status line, which includes the status code and reason message (e.g., `HTTP/1.1 200 OK`, which says the request was successful)
- Response header fields, such as `Content-Type: text/html`
- An empty line
- An optional message body

The following example shows a typical response:

```
HTTP/1.1 200 OK
Date: Thur, 21 May 2015 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2015 23:11:55 GMT
ETag: "xxxxxxxxxxxxxxxx"
Content-Type: text/html; charset=UTF-8
Content-Length: 131
Accept-Ranges: bytes
Connection: close

<!DOCTYPE html>
<html>
  <head>
    <title>I'm a useful title to this page</title>
  </head>
  <body>
    <p>I'm some interesting content that people can't wait to consume.</p>
  </body>
</html>
```

The status codes, such as 200 OK in the preceding example, provide details about the request. The most common types of status codes are the 4XX and 5XX codes. The 4XX codes are for client errors, with the most common being a 404, which denotes that the resource being requested does not exist. The 5XX codes are for server codes, the most common of which is 500, or an Internal Server error. Anyone who does much web development will quickly become accustomed to the dreaded 500 errors.

These verbs are needed because, by definition, HTTP is a stateless protocol. That is why nothing in the request identifies any previous request; instead, each request-response is expected to act completely independently.

Much of this communication happens behind the scenes and is handled by the user's browser and the web server. However, the information being sent and received affects your web application. As you continue developing your knowledge and skills about ASP.NET, you will find cases where

digging in deeper to the different values in either the request or the response becomes important. You may need to set request and/or response headers to ensure that some contextual information (such as an authorization token or the client's preferred language) are properly set.

Microsoft Internet Information Services

Microsoft Internet Information Services (IIS) is an application that comes with Microsoft Windows that is designed to support HTTP (known as a web server). It is included with all current versions of Windows, although it is not installed by default. When you develop an ASP.NET application, either Web Forms or MVC, the work of processing and creating the content that is returned to the client is done by IIS.

HTML 5

Whereas HTTP is the process of communicating between a client and a server, HTML is the core markup language of the Internet. HTML (HyperText Markup Language) is used for structuring and presenting content on the Web, and is a standard from the W3C (World Wide Web Consortium). HTML 5, finalized in October 2014, is the newest version of this standard. The previous version, HTML 4, was standardized in 1997.

As you can imagine, the Web went through some dramatic evolution during the 17 years between HTML 4 and HTML 5. While this evolution provided some advantages, especially to users, it also created some problems for website developers. One of the primary problems was that web browser companies tried to differentiate their products by providing a set of browser-specific enhancements, especially related to multimedia. This made developing an interactive website problematic because each browser had different specific development requirements.

HTML 5 was designed to help solve the problems caused by this fragmentation. Improvements include the following:

- Additional support for multimedia, including layout, video, and audio tags
- Support for additional graphics formats
- Added accessibility attributes that help differently abled users access the web page content
- Significant improvements in the scripting APIs that allow the HTML elements to interact with JavaScript (you will learn more about this in Chapter 14, "jQuery")

HTML Markup

HTML documents are human-readable documents that use HTML elements to provide structure to information. This structure is used to provide context to the information being displayed. A web browser takes the context and content into account and displays the information accordingly. These elements can be nested, meaning one element can be completely contained within another element, making the whole page basically a set of nested elements, as shown here:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>I'm a useful title to this page</title>
  </head>
  <body>
    <p>I'm some interesting content that people can't wait to consume.</p>
  </body>
</html>
```

Each layer of elements acts to group related content. As each element is parsed by the browser, it is evaluated as to where it belongs within the logical structure. This structure is what gives the browser the capability to relate content based upon its proximity and relationship to other elements within the structure. In the preceding example, the `title` element is a child of the `head` element.

Note also the expectation of both open and close tags. This fits in with the concept that an element can be contained within other elements. Only those elements that cannot contain other elements do not need to be closed. The open tag is a set of angled brackets `<>` around an element `html`, while the close tag is a set of angled brackets `</>` around the same `element` name but prefaced with a slash `/html`. This enables the browser to identify each section appropriately. Some browsers may support some tags that are not properly closed, but this behavior is inconsistent, thus care should be taken to close all elements. The only item that does not follow this standard is the `<!DOCTYPE html>` declaration. Instead, this identifies how the content that follows should be defined. In this case, the content is defined as `html`, so the browser knows that the content should be parsed as HTML 5.

Some of the more useful elements available in HTML 5 are listed in Table 1-2. This is not a complete list! Visit the W3C site for a complete list of HTML elements and a full description of their usage at `http://www.w3.org/TR/html5/index.html`.

TABLE 1-2: Commonly Used HTML Elements

ELEMENT NAME	DESCRIPTION
html	Identifies the content as HTML code.
head	Defines the content as the head section of the page. This is a high-level section containing information that the browser uses to control the display of the content.
title	An item within the head section, this element contains the content normally displayed in the browser's title bar.
body	Defines the content as the body section of the page. This section contains the content that is displayed within the browser window.
a	Anchor tag that acts as a navigation link to other content. It can redirect the user to another location on that same page or to a completely different page.
img	This tag places an image onto the page. It is one of the few elements that does not have a closing tag.

ELEMENT NAME	DESCRIPTION
form	The `form` tag identifies the contained content as a set of information that will be submitted together as a block. It is generally used to transfer information from the user to the server.
input	This element plays a lot of roles within a form. Depending upon the type (much more on this later!) it can be a text box, a radio button, or even a button.
span	A way to delimit content inline. This enables you to give a special format to one or more words in a sentence without affecting the spacing of those words.
div	Like the `span` tag, this tag acts as a container for content. However, it is a block element, and different in that there is a line break before and after the content.
audio	An HTML 5 feature that allows you to embed an audio file into the page. The types of audio files supported may differ according to browser.
video	An HTML 5 feature to embed video files into the page so that the browser will play the content inline.
section	An HTML 5 addition that identifies a set of content as belonging together. Think of it as a chapter in a book, or areas of a single web page such as introduction and news.
article	Another HTML 5 addition that defines a more complete, self-contained set of content than the section element.
p	A paragraph element that breaks up content into related, manageable chunks.
header	Provides introductory content for another element, generally the nearest element. This may include the body, which means the content is the header for the entire page.
h1, h2, h3	An element that enables content to be designated as header text. The smaller the number, the higher it appears in the hierarchy. An `h1` element would be similar to a book title, `h2` could be chapter title, `h3` section title, and so on.
ul	Enables the creation of an unordered, bulleted list.
ol	Enables the creation of an ordered, generally numbered list.
li	The list item element tells the browser that the content is one of the items that should be included in a list.

Attributes in HTML

An attribute is additional information that is placed within the angle braces of the opening element. This attribute provides detail so that the browser knows how to behave when rendering or

interacting with that element. An example is the anchor element, which provides a navigational link to other content:

```
<a href='http://www.wrox.com'>Awesome books here!</a>
```

The `href` is an attribute that tells the browser where to send users after they click the "Awesome books here!" link.

All elements support attributes of some sort, whether they are implied required items such as the `href` attribute in an anchor tag, or optional tags such as `name`, `style`, or `class`, which can be used to control the identification and appearance of the attributed element.

HTML Example

The code in Listing 1-1 is a sample HTML page that contains almost all of the elements in Table 1-2.

LISTING 1-1: An example HTML page

```
<!DOCTYPE html>
<html>
    <head>
        <title>Beginning ASP.NET Web Forms and MVC</title>
    </head>
    <body>
       <!-- This is an HTML comment.  The video and audio elements are not
displayed.-->
       <article>
         <header>
            <h1>ASP.NET from Wrox</h1>
            <p>Creating awesome output</p>
            <a href='http://www.wrox.com'>
               <img src='http://media.wiley.com/assets/253/59/wrox_logo.gif'
                  width='338' height='79' border='0'>
            </a>
         </header>
         <section>
            <h2>ASP.NET Web Forms</h2>
            <p>More than a decade of experience and reliability.</p>
            <ol>
               <li>Lots of provided controls</li>
               <li>Thousands of examples available online</li>
            </ol>
         </section>
         <section>
            <h2>ASP.NET MVC</h2>
            A new framework that emphasizes a <div>stateless</div> approach.
            <ul>
               <li>Less page-centric</li>
               <li>More content centric</li>
            </ul>
         </section>
```

```
        </article>
        <form>
           <p>
              Enter your <span style='color: purple'>email</span> to sign up:
              <input type='text' name='emailaddress'>
           </p>
           <input type='submit' value='Save Email'>
        </form>
     </body>
  </html>
```

Microsoft's Internet Explorer renders this HTML content as shown in Figure 1-2. All other HTML 5 browsers will also render this comment in a very similar way.

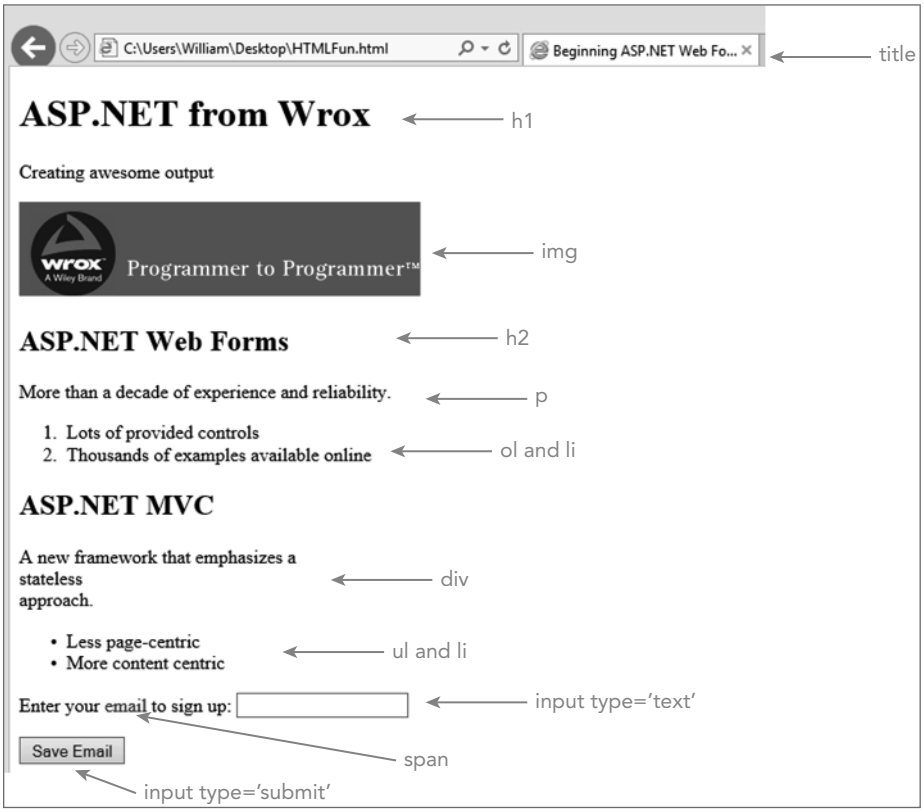

FIGURE 1-2: HTML rendered in the browser

As you can see, HTML provides some simple layout to the content. However, when you look at various sites on the Web, you will likely not see anything that looks like the preceding example. That's because HTML provides layout, but there is another technology that provides more control over the user experience (UX) by enhancing design. This technology is Cascading Style Sheets (CSS).

> ***REFERENCE*** *CSS is explained in more detail in Chapter 3, "Designing Your Web Pages."*

ASP.NET Web Forms

ASP.NET Web Forms have been part of the .NET infrastructure since its initial release. Web Forms generally take a page-based approach whereby each web page that may be requested is its own unique entity. During development there are two physical pages in the file system that make up each viewable page: the `.aspx` code, which contains the viewable markup, and the `.aspx.cs` or `aspx.vb`, which contains the code to do the actual processing, such as creating the initial content or responding to button clicks. These two pages together provide the code and markup necessary to create the HTML that is sent to the browser for viewing.

The main benefit of ASP.NET Web Forms is the level of abstraction that it provides compared to the request/response approach and the creation of the HTML that is sent to the client. A detailed knowledge of HTML is less critical than a detailed knowledge of C# or Visual Basic. The framework itself hides a lot of the HTML generation by doing it for you.

The primary model for communications between the client and the server is an approach called the *postback*, whereby a page is rendered in the browser, the user takes some action, and that page is sent back to the server using the same resource name. This allows each page to be responsible for both the creation of the page content and responding to changes in the page content as necessary.

ViewState

This response to change in the page content is enhanced through the use of ViewState. Because HTTP is a stateless protocol, anything that needs state needs to be managed in a more customized approach. ViewState is how ASP.NET Web Forms take this customized approach and transfer state information between the browser and the server. ViewState is a hidden field `<input type="hidden" name="_VIEWSTATE" value="blah blah">` that is included within the page. The entity's value contains hashed information that is unreadable by humans. Fortunately, ASP.NET is able to parse the information and get an understanding of the previous version of the various items on the page.

It is important to understand view state because of the significant role it plays in how ASP.NET Web Forms do their work. Say you are working on a page that has several postbacks. Perhaps one of the postbacks changes the value of a label. If the label had a default value from the first rendering, every initialization of that control on each new postback will reset that value to the default value. However, the system then analyzes the view state and determines that this particular label has a different value that should be displayed. The system now recognizes that it is in a different state and will override the default setting to set the label to the newer, changed version of the text.

This is a powerful way to persist changes between multiple postbacks. However, the more items that change and need to be tracked, the larger the set of view state information, which can be

problematic. This information is passed both directions, from server to the client, and then back to the server. In some cases the amount of information being transferred as part of the view state can slow down the download/upload time, especially in those cases where network speed or bandwidth is limited.

By default, the use of ViewState is enabled on every control. However, as the developer you can override those settings as necessary, such as when you know that you won't need to know the previous state of the control. You can also use the view state programmatically. Imagine a large list of data that has both sorting capabilities and paging. If you are going to sort before paging, then the sorting criteria needs to be stored somewhere so that it is available to the next postback. The view state is one place to store this information.

ASP.NET Web Forms Events and Page Lifecycle

One of the strengths of Web Forms is the ability it gives developers to plug into the various events in the page lifecycle. The ASP.NET lifecycle allows the developer to interact with information at various points in the HTML creation phase. As part of the flow, the developer can also use event handlers to respond to events that may happen on the client, including clicking a button or selecting an item in a dropdown list. For developers who are coming from a traditional event-driven development approach, such as Windows Forms, this approach will be very easy to pick up. While the lifecycle process gives a lot of power to a developer, it also adds to the complexity of the application—the same code can result in a different outcome depending on when it is called during the lifecycle.

The steps in the lifecycle are shown in Table 1-3. Some of these items may not make any sense to you at this point, but as we move through the process of creating an interactive web site, you will start to see how this all comes together.

TABLE 1-3: ASP.NET Page Lifecycle Stages

STAGE	DESCRIPTION
Request	This stage happens before the page-calling process starts. It is when the system determines whether run-time compilation is necessary, whether cached output can be returned, or whether a compiled page needs to be run. There are no hooks into this stage from within the ASP.NET page itself.
Start	The page starts to do some processing on the HTTP request. Some base variables are initialized, such as `Request`, `Response`, and the `UICulture`. The page also determines if it is a postback.
Initialization	During this phase, the controls on the page are initialized and assigned their unique IDs. Master pages and themes are applied as applicable. None of the postback data is available, and information in the view state has not yet been applied.
Load	If the request is a postback, control information is loaded with the information recovered from view state.

continues

TABLE 1-3 *(continued)*

STAGE	DESCRIPTION
Postback event handling	If the request is a postback, all the various controls fire their event handlers as needed. Validation also happens at this time.
Rendering	Before the rendering stage starts, ViewState is saved for the page and all of the controls as configured. At this time, the page output is added to the response so that information may start flowing to the client.
Unload	This happens after the content was created and sent to the client. Objects are unloaded from memory and cleanup happens.

The steps in the lifecycle are exposed through a set of lifecycle events. A developer can interact with a lifecycle event as necessary. You will learn more about this interaction as you develop the sample application. These events are listed in Table 1-4.

TABLE 1-4: Lifecycle Events for ASP.NET Pages

EVENT	DESCRIPTION AND TYPICAL USE
Preinit	Raised after the start stage is complete and before the initialization stage begins. Typically used to create or recreate dynamic controls, setting master pages or themes dynamically (more on this later). Information in this stage has not yet been replaced with the ViewState information, covered earlier.
Init	This event is raised after all the controls have been initialized. It is typically used to initialize control properties. These initializations do not affect view state.
InitComplete	Only one thing happens between `Init` and `InitComplete`, and that is the enabling of view state for the controls. Changes applied in this event and after will impact view state, so are available upon postback.
PreLoad	Raised after the page manages the view state information for itself and all controls. Postback data is also processed.
Load	The `OnLoad` method is called in a page, which then recursively calls that same method on every control. This is typically where the majority of your creation work happens, initializing database connections, setting control values, etc.
Control Events	These are specific control-based events, such as the `Click` of a button, or `TextChanged` on a text box.

EVENT	DESCRIPTION AND TYPICAL USE
LoadComplete	This event is raised after all the event handling has occurred. Doing anything here would generally require all of the controls to be loaded and completed.
PreRender	After all the controls have been loaded, the `Page` object starts its Prerender phase. This is the last point at which you can make any changes to the content or the page.
PreRenderComplete	Raised after every databound control has been bound. This happens at the individual control level.
SaveStateComplete	Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.
Render	This is not an event. Rather, at this point in the process, the `Page` object calls this method on each control. All ASP.NET Web server controls have a `Render` method that writes out the control's markup to send to the browser.
Unload	This is used to perform special cleanup activities, such as closing file or database connections, logging, etc.

The work that you will be doing in the sample application only takes advantage of a few of these events. However, understanding that they may occur gives you an idea of how ASP.NET Web Forms works under the covers. Web Forms enable you to tap into each of these events as needed, both at a page level and a control level. While you will likely encounter entire application projects that don't require anything outside of the Load and Control Events sections, Web Forms provide you with the power to do so as needed.

Some of the more powerful controls have their own sets of events, which you will learn about when you start to work on the sample application.

Control Library

One of the benefits of ASP.NET Web Forms is a powerful set of built-in server controls that give developers a boost in development speed and enhance rapid application development (RAD). Using these controls turns the development process into one that's more about configuration than development, providing an out-of-the-box experience that will likely satisfy many developers who need the most common default behavior. In addition, because of the maturity of this approach, an extensive set of third-party controls are available as well as rich and powerful support within Visual Studio.

These ASP.NET server controls are items that a developer places on an ASP.NET web page. They run when the page is requested, and their main responsibility is to create and render markup to the

browser. Many of these server controls are similar to the familiar HTML elements, such as buttons and text boxes. Other of these server controls allow for more complex behavior, such as a calendar control that manages the display of data in a calendar format and other controls that you can use to connect to data sources and display data:

There are four main types of controls:

- HTML server controls
- Web server controls
- Validation controls
- User controls

HTML Server Controls

HTML server controls are generally wrappers for traditional HTML elements. These wrappers enable the developer to set values in code and to use events, such as a textbox control firing an event when its text display value has been changed. You will be working with many different HTML server controls as you work through the Web Forms part of the application.

Web Server Controls

A web server control acts as more than a wrapper around an HTML element. It tends to encompass more functionality and be more abstract than an HTML server control, because it does more things. A calendar control is a good example of a web server control; it enhances UI functionality by providing a button that enables users to access a grid-like calendar to select the appropriate date. The calendar control also provides other functionality, such as limiting the range of selectable dates, formatting the date being displayed, and moving through the calendar by month or year.

Validation Controls

The third type of control is the validation control. This control ensures that the values entered into other controls meet certain criteria, or are valid. A textbox that is expected to only capture money amounts, for example, should only accept numbers and perhaps the comma (,) and period (.). It should also ensure that if the value entered contains a period, then there are no more than two numbers to the right of the period. The validator provides this support on the client side and on the server. This ensures that the data is correct before being sent to the server and then ensures that the data is correct when it gets to the server.

User Controls

The last type of control is a user control. This is a control that you build yourself. If a set of functionality needs to be available on multiple pages, then it is most likely that you should create this functionality as a user control. This enables the same control to be reused in multiple places, rather than copying the code itself into multiple pages.

These controls can do a lot of very useful things for you, but they come at a cost. By using these controls, you may lose some control over the finished HTML, which may lead to bloated output or HTML that does not quite fit what the designer may desire.

ASP.NET MVC

Earlier, you learned that ASP.NET Web Forms is a page-based approach to designing a web application. ASP.NET MVC is a different architectural approach that emphasizes the separation of concerns. Whereas Web Forms are generally made up of two sections, markup and code-behind, MVC breaks the concerns into three parts, model, view, and controller. The model is the data that is being displayed, the view is how the data is being displayed to the user, and the controller is the intermediary that does the work of ensuring that the appropriate model is presented to the correct view. Figure 1-3 illustrates the interaction between the different parts.

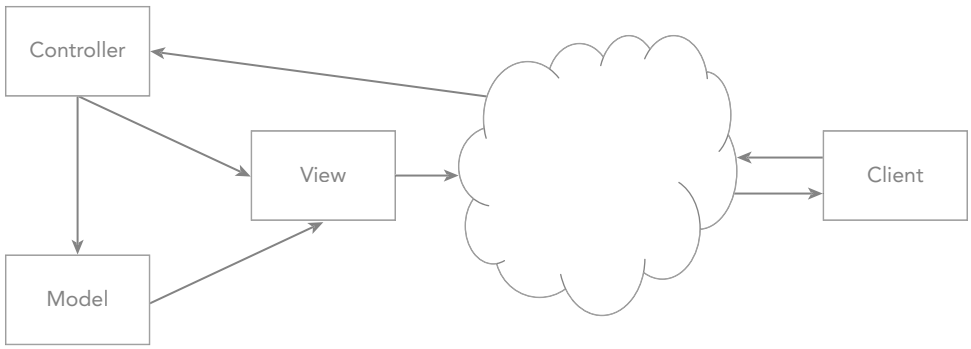

FIGURE 1-3: Model-View-Controller (MVC) design

A key difference between ASP.NET Web Forms and MVC is that MVC presents views, not pages, to the client. This is more than simple semantics, it indicates a difference in approach. Web Forms take a file system approach to presenting content; MVC takes an approach whereby content is based on the "type of action" that you are trying to perform on a particular thing, as shown in Figure 1-4.

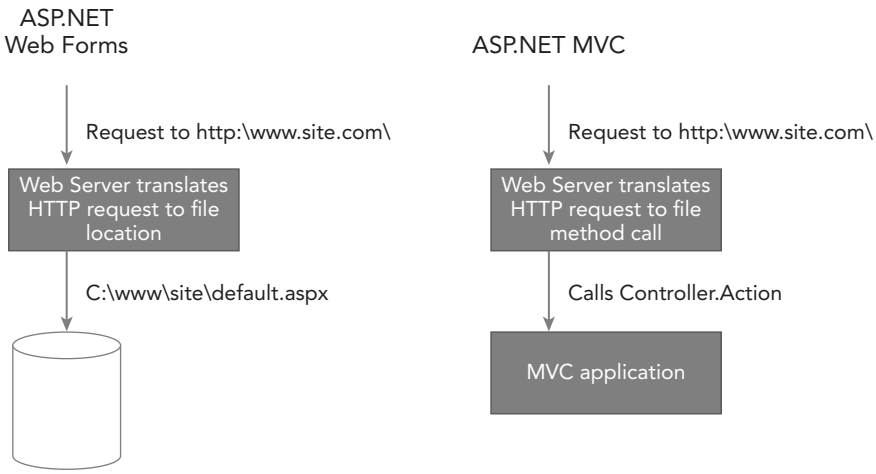

FIGURE 1-4: Different Approaches Between MVC and Web Forms

> ***NOTE*** *This kind of approach may be less intuitive for developers who are coming from a more event-driven background. However, developers who have experience with other MVC approaches, such as Ruby on Rails, will find the MVC pattern to be comfortable and a good fit with their previous experience.*

The key reason for the MVC pattern's success is the degree to which it helps developers create applications whose different aspects can be separated (input logic, business logic, and UI logic), while still providing a relatively loose coupling between these elements. A loosely coupled system is one in which each component has very little to no knowledge of the other components. This enables you to make changes in one of the components without disturbing the others.

In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles query-string values, and passes these values to the model, which in turn might use these values to query the database. Because of this separation, you can completely redesign the UI without affecting the controller or model at all. Because of the loose coupling, the interdependency is much less rigorous. It also enables different people to assume different roles in the development of the application, by disassociating the HTML creation from the server that creates the data to be displayed.

The MVC pattern specifies where each type of logic should be located within your application. The UI-specific logic belongs in the view. Input logic, or the logic that handles the request from the client, belongs in the controller. Business logic belongs in the model layer. This separation helps you manage complexity when you build an application because it enables you to focus on one aspect of the implementation at a time.

Testability

An important consideration when using an MVC approach is the valuable increase in testability it offers. Unit tests are re-runnable items that validate a particular subset of functionality. This is important in modern development because these unit tests enable the developer to refactor, or make changes to, existing code. The unit tests enable developers to determine whether any negative side effects result from the change by running the already created unit tests. An ASP.NET Web Forms application is difficult to unit test for precisely the same reasons that it works so well as a RAD approach: the power of the built-in controls and the page lifecycle. They are very specific to the page of which they are a part, so trying to test discrete pieces of functionality becomes much more complicated because of the dependencies with other items on the page.

ASP.NET MVC's approach and separation means that controllers and models can be fully tested. This ensures that the behavior of the application can be better evaluated, understood, and verified. When building a very simple application this may not be important, but in a larger, enterprise-level application it becomes critical. The functionality it provides to the business might be essential, and it will likely be managed, maintained, tweaked, and changed over a long lifetime; and the more complex the code, the more risk that a change in one area may impact other areas. Running unit tests after a set of changes provides assurance that previously created functionality continues to work as expected. Building unit tests on new functionality verifies that the code is working as expected and provides insurance against future changes.

You won't be specifically building unit tests as part of the process in building the sample web application. However, the available source code does have a unit test project and some tests will be created as you work through the development process, especially for those areas that are using ASP .NET MVC.

Full Control over Output

ASP.NET MVC does not have the same dependence upon controls that ASP.NET Web Forms do, thus it does not have the same risk of becoming bloated HTML output. Instead, developers create the specific HTML that they want sent to the client. This allows full access to all attributes within an HTML element rather than just those allowed by the ASP.NET Web Form server control. It also allows for much more predictable and clearly understood output. Another advantage in having full control over the rendered HTML is that it makes the inclusion of JavaScript much easier. There is no potential for clashes between control-created JavaScript and developer-created JavaScript; and because the developer controls everything that is rendered on the page, using element names and other attributes that may have been commandeered by the generated HTML becomes easier.

Of course, this additional flexibility comes at some cost: Developers are required to spend more time building the HTML than otherwise may have been necessary with the Web Form controls. It also requires that developers be more knowledgeable about HTML and client-side coding, such as JavaScript, than was necessary with Web Forms.

Web Forms and MVC Similarities

It is important to understand that Web Forms and MVC are not opposing approaches but rather different approaches that have inherently different strengths and weaknesses. They each address different concerns and are not mutually exclusive. A developer can create unit tests in Web Forms; it just takes more work and requires the developer to add abstraction where the framework does not provide any by default. Just as with virtually any other development problem, there are multiple potential solutions and approaches. A well-designed application will be successful, regardless of the approach taken.

Fundamentally, as both Web Forms and MVC are designed to solve the same base requirement—creating HTML content that will be provided to the client user—there are a lot of similarities between the two. Properly architected applications will be much the same, especially in terms of backend processing. Accessing databases, web services, or file system objects will all be the same regardless of approach. This is why many developers can become proficient in both.

Choosing the Best Approach

As described earlier, each of these frameworks has its own set of advantages and disadvantages. You need to evaluate your requirements against these concerns and determine which is the most important to your project. This means that there is no right answer; some projects would be best implemented via Web Forms, whereas others might be better served by taking an MVC approach.

There are additional concerns when determining the appropriate development approach, including the background and experience of the developers who will be doing the work and how much information is being shown the same way on multiple pages.

Fortunately, with the advent of Visual Studio 2015 and ASP.NET 5.0 you no longer have to make an either/or choice. With a little bit of maneuvering, you can create a project that uses both approaches as necessary, enabling you to determine on a case-by-case basis which approach to use, instead of using a site-by-site determination.

This case-by-case approach is used in the sample application, which uses both ASP.NET Web Forms and ASP.NET MVC to solve various business problems presented.

USING VISUAL STUDIO 2015

Microsoft's Visual Studio is the primary integrated development environment (IDE) used to create ASP.NET sites and applications. The most recent version is Microsoft Visual Studio 2015, which includes quite a few enhancements. There are also new versions of both C#, version 6.0, and VB.NET, version 14. ASP.NET 5 is also an important release because it can now run on OS X and Linux with Mono installed.

Mono is a software platform designed to enable developers to easily create cross-platform applications. It is an open-source implementation of Microsoft's .NET Framework that runs on non-Windows operating systems. This is a tremendous game changer; because until now, every ASP.NET application, either Web Form or MVC, needed to be deployed to and run on a Microsoft Windows server.

Versions

Several different versions of Visual Studio are available for web developers:

- **Visual Studio Community Edition:** A free version of Visual Studio that is designed to help hobbyists, students, and other non-professional software developers build Microsoft-based applications
- **Visual Studio Web Developer Express:** Another free version of Visual Studio, supporting only the development of ASP.NET applications
- **Visual Studio Professional Edition:** A full IDE for use in creating solutions for the Web, desktop, server, cloud, and phone
- **Visual Studio Test Professional Edition:** Contains all the features of the Professional Edition, with the capability to manage test plans and create virtual testing labs
- **Visual Studio Premium Edition:** Contains all the features of the Professional Editions with the addition of architect-level functionality related to analyzing code and reporting on unit testing and other advanced features
- **Visual Studio Ultimate Edition:** The most complete version of Visual Studio, including everything needed for development, analysis, and software testing

The sample application will use the Community Edition because it provides a complete Visual Studio experience.

Downloading and Installing

Downloading and installing Visual Studio is straightforward. The following Try It Out takes you through the various steps involved, from downloading the correct edition, to selecting appropriate options, and completing the install.

TRY IT OUT Installing Visual Studio

1. Go to `http://www.visualstudio.com/products/visual-studio-community-vs`. You will see a site similar to what is shown in Figure 1-5.

FIGURE 1-5: Visual Studio site to download Community Edition

2. Select the green Download button to run the installation program. Running the download will give you the screen shown in Figure 1-6.

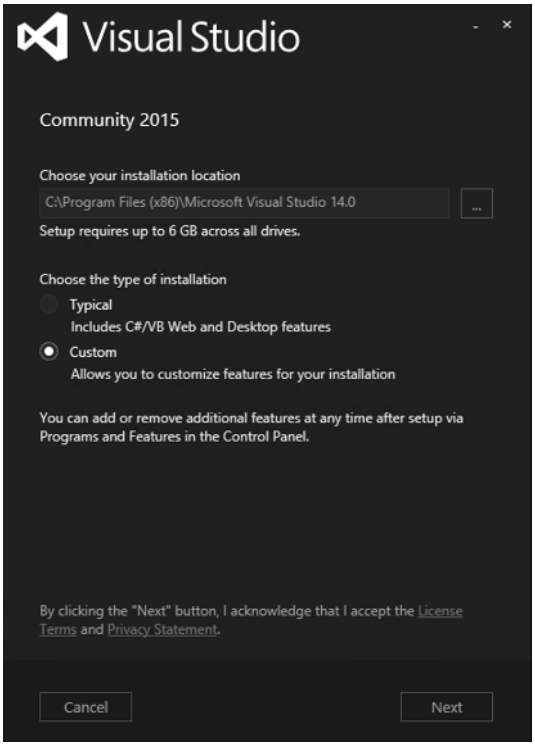

FIGURE 1-6: Installation screen for Community Edition

3. You can select the Custom radio button and see the screen as shown in Figure 1-7, or you can choose Typical and start the installation process.
4. Leave the default settings and click the Install button. You will likely get a User Account Control acceptance box to which you must agree before continuing, after which the download and installation process begins. This may take a while. When the installation is completed you will see a window like the one shown in Figure 1-8. Once completed you may need to restart your computer.
5. To launch the application, click the Launch button. This will bring you to the login screen shown in Figure 1-9.
6. For now, skip the login. This will bring up the Development Settings and Color Theme selection screen shown in Figure 1-10.

FIGURE 1-7: Select items to install

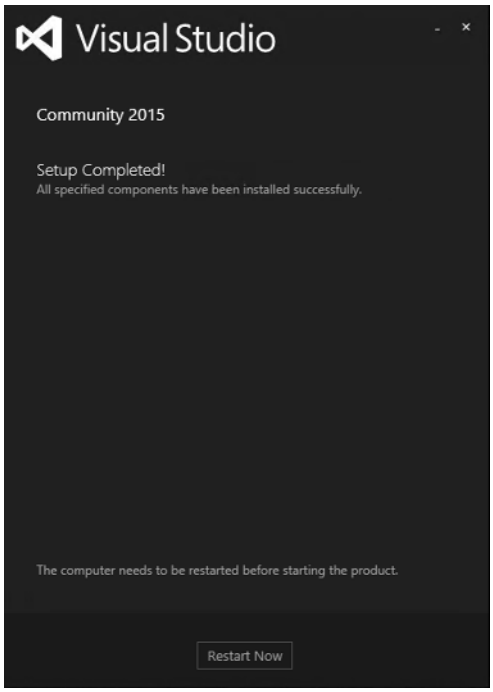

FIGURE 1-8: Setup Completed window

FIGURE 1-9: Login screen in Visual Studio

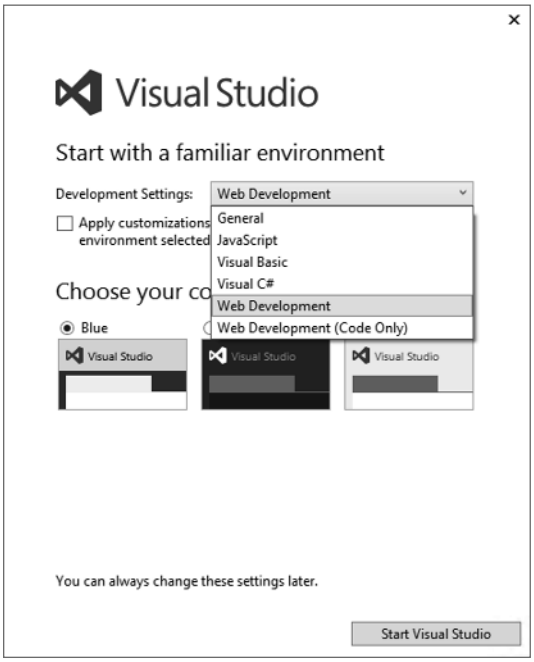

FIGURE 1-10: Initial configuration of Visual Studio

7. Select the Web Development option, and whichever set of colors you prefer. After configuring these preferences, the application will open, as shown in Figure 1-11.

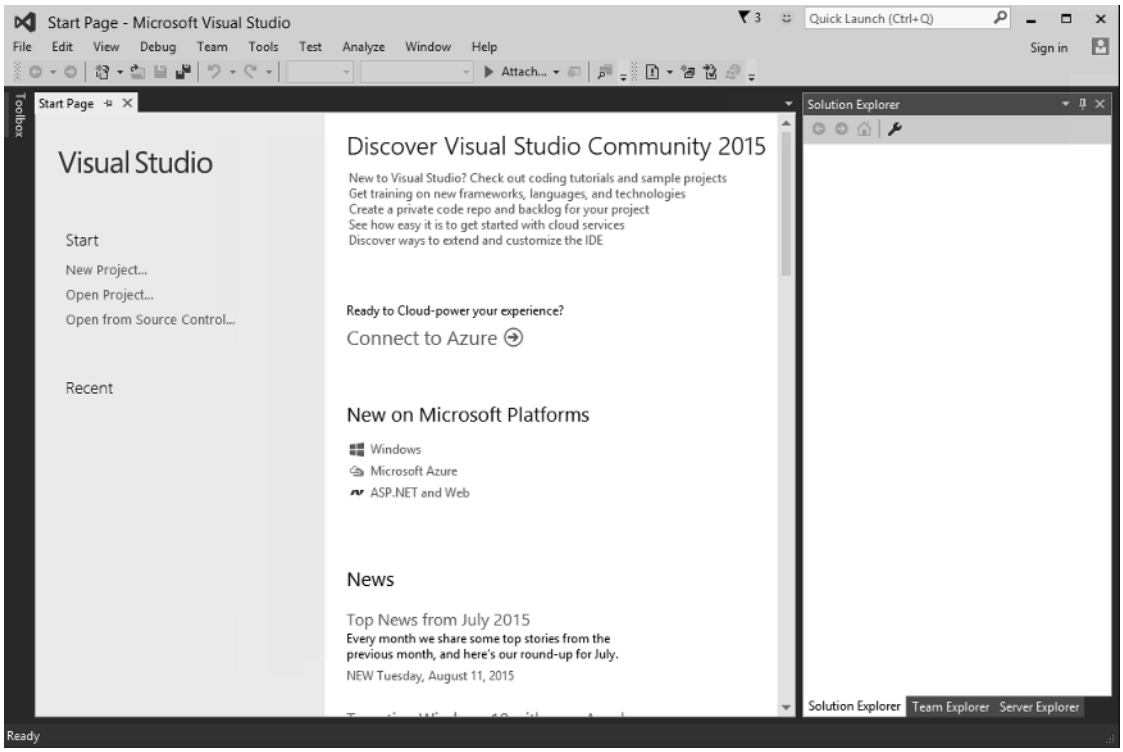

FIGURE 1-11: Start Page for Visual Studio

How It Works

You have completed installing Visual Studio. It is a relatively straightforward installation process, with the only unusual aspect being that Visual Studio now gives you the opportunity to link your installation to an online profile. This enables you to share source code repository information and some system settings between different installations of Visual Studio

If you have not used Visual Studio before don't worry; you will be spending a lot of time going through it as you build the sample application.

THE SAMPLE APPLICATION

The best way to learn how to do something, such as build an Internet application, is simply to do it. With this in mind, you will be building a real application as we go through each functional area of ASP.NET. We will be developing an application called RentMyWrox that acts as a loaning library.

Because this app supports both ASP.NET Web Forms and ASP.NET MVC, there will be some duplication of code and/or effort to show critical features in both frameworks. For some functionality you will be able to do this in two different pages; with other functionality you will have to replicate the same functionality both ways, basically replacing one version with the other version.

The requirements for this application are as follows:

- The site owner (administrator) can create a list of items that are available for rent or borrowing.
- The items contain pictures and text.
- Users can create and register an account online that will give them secure access to the application.
- Users can log in and select one to many items that they want to check out.
- The listing of items can be filtered.
- Users can complete their reservation through a type of checkout process.

These requirements will give you the opportunity to go over the design of the look and feel of the website, getting and saving information in a database, and handling user account creation and authentication using both ASP.NET Web Forms and MVC approaches.

SUMMARY

Microsoft has provided many different web application frameworks over the years. Before the .NET Framework was introduced, there was an approach that provided the capability to incorporate HTML5 markup with business processing. This approach, now known as "Classic ASP," was innovative at the time and enabled developers to quickly and relatively easily build complex business applications.

ASP.NET follows in those footsteps by providing developers with a framework on which to balance all development work. When ASP.NET was introduced, only a single development framework was supported: ASP.NET Web Forms. This framework took a page-bound approach, tying together a specific page and a resource name that would be called. There were two physical pages to each resource page: one page containing the HTML markup that would be returned to the client, and another page that provided all the processing. This allowed for a separation of concerns that Classic ASP did not address.

However, after several years Microsoft released another framework: ASP.NET MVC. This approach allows even more separation of concerns, and greatly enhances the capability to test the business processing in an automated fashion. This is important because it dramatically increases confidence in the correctness of the code.

All of these frameworks are designed to do one single thing: provide HTML from the server to a client. HTML is the language of the Internet—it incorporates the layout and markup of everything that you would see on a web site. The creation of this HTML is primary. Obviously, other

processing is going on in the background, but every representation of this work back to the requesting client will be HTML.

EXERCISES

1. What is the difference between HTML and HTTP?
2. Why is ViewState important when you are working with ASP.NET Web Forms?
3. What are the three different architectural components of ASP.NET MVC?
4. What is Microsoft Visual Studio and what are we using it for in this book?

▶ WHAT YOU LEARNED IN THIS CHAPTER

Attributes	Extra information you can put in an HTML element that may change how that element interacts with the browser or with the user.
Elements	A section of HTML that defines a set of content. The elements define the content because there is an opening tag `<p>` and a closing tag `</p>` around the content.
HTML	Hypertext Markup Language, how content is identified on the Internet so that browsers know how to handle and display the information.
HTTP	Hypertext Transfer Protocol, the definition that handles the request/response behavior which delivers information from the client to the server and back.
IDE	Integrated development environment, a collection of tools and aids that help developers build programs and applications.
MVC	An architectural pattern that separates the responsibilities of a website into three different components: models, views, and controllers. Each of the sections takes responsibility for part of the process of building a user interface.
Web Forms	An approach to building web applications that is based on a page approach, so each set of functionality is its own page, responsible for both its rendering and business logic.