ASP.NET Page Structure

- ASP.NET pages are simply text files that have the `.aspx` file name extension, and can be placed on any web server equipped with ASP.NET.
- When a client requests an ASP.NET page, the web server passes the page to the ASP.NET runtime, a program that runs on the web server that's responsible for reading the page and compiling it into a .NET class. This class is then used to produce the HTML that's sent back to the user. Each subsequent request for this page avoids the compilation process: the .NET class can respond directly to the request, producing the page's HTML and sending it to the client, until such time as the .aspx file changes. This process is illustrated in Figure 2.1.
- An ASP.NET page consists of the following elements:

  - directives
  - code declaration blocks
  - code render blocks
  - Web form
  - Web Server Controls

- Figure 2.2 illustrates the various parts of a simple ASP.NET page.

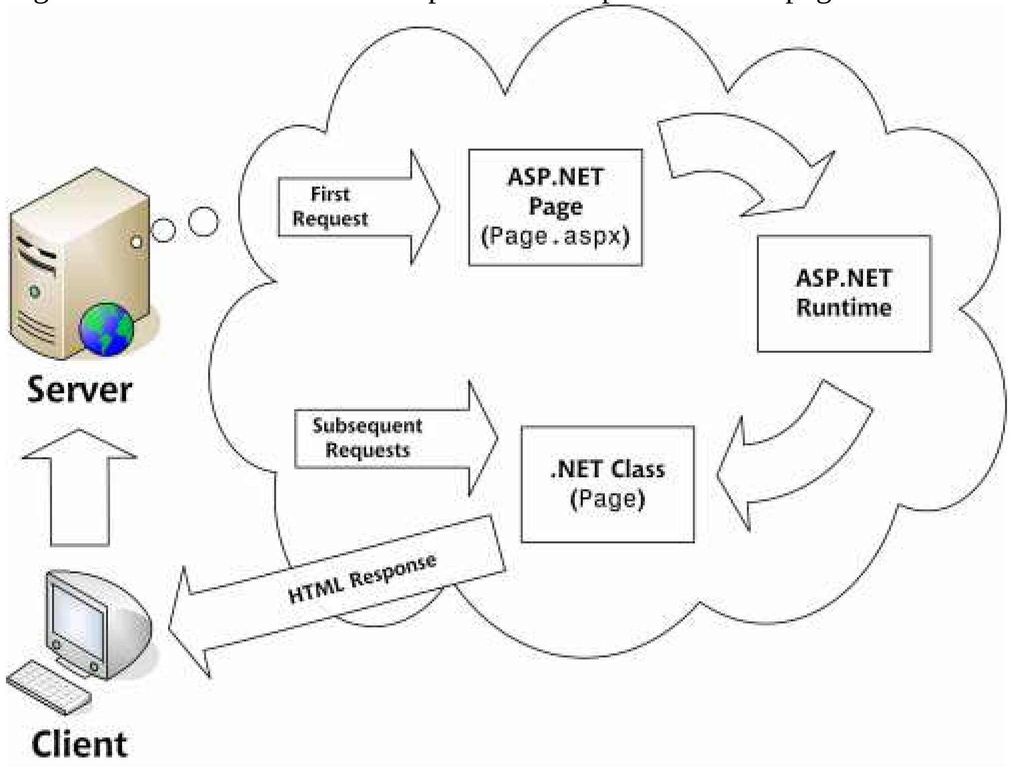
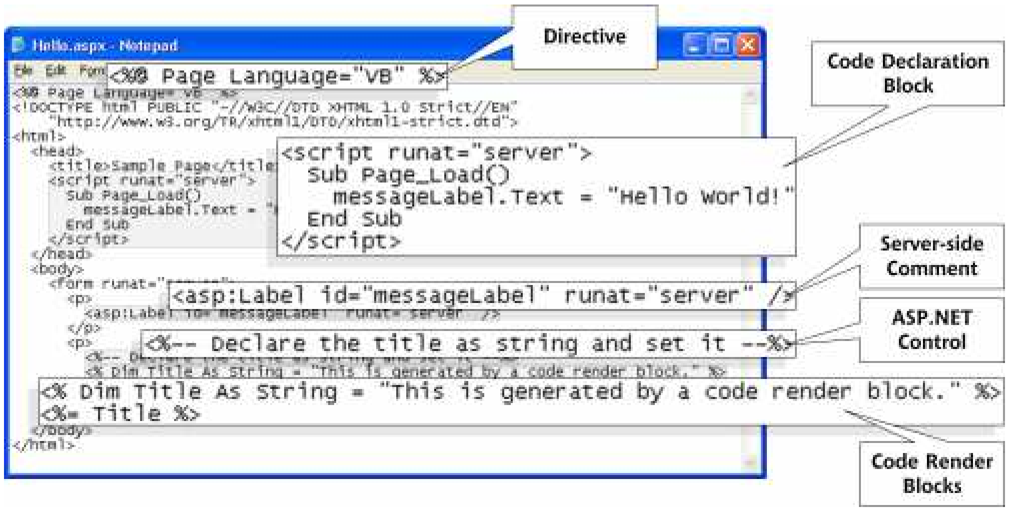


Figure 2.1. The life cycle of the ASP.NET page



```
<%@ Page Language="VB" %>
```

```
<script runat="server">
  Sub Page_Load()
    messageLabel.Text = "Hello World!"
  End Sub
</script>
```

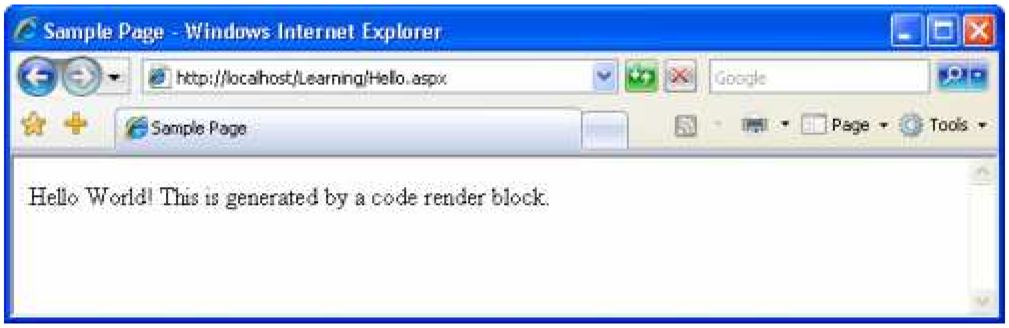
```
<asp:Label id="messageLabel" runat="server" />
```

```
<%-- Declare the title as string and set it --%>
```

```
<% Dim Title As String = "This is generated by a code render block." %>
<%= Title %>
```

Figure 2.2. The parts of an ASP.NET page

To make sure we're on the same page and that the code works, save this piece of code in a file named `Hello.aspx` within the Learning virtual directory you created in Chapter 1. Loading the file through `http://localhost/Learning/Hello.aspx` should render the result shown in Figure 2.3.


Sample Page - Windows Internet Explorer
http://localhost/Learning/Hello.aspx
Google
Sample Page
Page
Tools

Hello World! This is generated by a code render block.

Figure 2.3. Sample Page in action

As you can see, this ASP.NET page contains examples of all the above components (except server-side includes) that make up an ASP.NET page. You won't often use every single element in a given page, but it's important that you are familiar with these elements, their purposes, and how and when it's appropriate to use them.

❖ ***Directives***

The directives section is one of the most important parts of an ASP.NET page. Directives control how a page is compiled, specify how a page is cached by web browsers, aid debugging (error-fixing), and allow you to import classes to use within your page's code. Each directive starts with `<%@`. This is followed by the directive name, plus any attributes and their corresponding values. The directive then ends with `%>`.

There are many directives that you can use within your pages, and we'll discuss them in greater detail later, but, for now, know that the Import and Page directives are the most useful for ASP.NET development. Looking at the sample ASP.NET page in Figure 2.2, we can see that a `Page` directive was used at the top of the page like so:

`Example 2.1. Hello.aspx (excerpt)`

```
<%@ Page Language="VB" %>
```

`Example 2.2. Hello.aspx (excerpt)`

```
<%@ Page Language="C#" %>
```

In this case, the `Page` directive specifies the language that's to be used for the application logic by setting the Language attribute. The value provided for this attribute, which appears in quotes, specifies that we're using either VB or C#. A whole range of different directives is available; we'll see a few more later in this chapter.

Unlike ASP, ASP.NET directives can appear anywhere on a page, but they're commonly included at its very beginning.

❖ ***Code Declaration Blocks***

In Chapter 3, VB and C# Programming Basics, we'll talk about code-behind pages and how they let us separate our application logic from an ASP.NET page's HTML. However, if you're not working with code-behind pages, you must use code declaration blocks to contain all the application logic of your ASP.NET page. This application logic defines variables, subroutines, functions, and more. In our page, we've placed the code inside `<script>` tags, like so:

```
<script runat="server">
 Sub mySub()
   ' Code here
 End Sub
</script>
```

Here, the tags enclose VB code, but it could just as easily be C#:

```
<script runat="server">
 void mySub()
 {
   // Code here
 }
</script>
```

Comments in VB and C# Code

Both of these code snippets contain comments--explanatory text that will be ignored by ASP.NET, but which serves to describe to us how the code works.

In VB code, a single quote or apostrophe (`'`) indicates that the remainder of the line is to be ignored as a comment.

In C# code, two slashes (`//`) achieve the same end. C# code also lets us span a comment over multiple lines if we begin it with `/*` and end it with `*/`, as in this example:

```
<script runat="server">
 void mySub()
 {
   /* Multi-line
      comment    */
 }
</script>
```

Code declaration blocks are generally placed inside the head of your ASP.NET page. The sample ASP.NET page shown in Figure 2.2, for instance, contains the following code declaration block:

Perhaps you can work out what the equivalent C# code would be:

```
Example 2.4. Hello.aspx (excerpt)

<script runat="server">
 void Page_Load()
 {
   messageLabel.Text = "Hello World";
 }
</script>
```

The `<script runat="server">` tag also accepts two other attributes. We can set the language that's used in this code declaration block via the language attribute:

```
<script runat="server" language="VB">

<script runat="server" language="C#">
```

If you don't specify a language within the code declaration block, the ASP.NET page will use the language provided by the `language` attribute of the `Page` directive. Each page's code must be written in a single language; for instance, it's not possible to mix VB and C# in the same page.

The second attribute that's available to us is `src`; this lets us specify an external code file for use within the ASP.NET page:

```
<script runat="server" language="VB" src="mycodefile.vb">

<script runat="server" language="C#" src="mycodefile.cs">
```

❖ ***Code Render Blocks***

If you've had experience with traditional ASP, you might recognize these blocks. You can use code render blocks to define inline code or expressions that will execute when a page is rendered. Code within a code render block is executed immediately when it is encountered--usually when the page is loaded or rendered. On the other hand, code within a code declaration block (within `<script>` tags) is executed only when it is called or triggered by user or page interactions. There are two types of code render blocks?inline code, and inline expressions--both of which are typically written within the body of the ASP.NET page.

Inline code render blocks execute one or more statements, and are placed directly inside a page's HTML between `<%` and `%>` delimiters. In our example, the following is a code render block:

```
Example 2.5. Hello.aspx (excerpt)
```

```
<% Dim Title As String = "This is generated by a code render " & _
   "block." %>
```

```
Example 2.6. Hello.aspx (excerpt)
```

```
<% string Title = "This is generated by a code render block."; %>
```

These code blocks simply declare a `String` variable called `Title`, and assign it the value `This is generated by a code render block`.

Inline expression render blocks can be compared to Response.Write in classic ASP. They start with `<%=` and end with `%>`, and are used to display the values of variables and methods on a page. In our example, an inline expression appears immediately after our inline code block:

```
Example 2.7. Hello.aspx (excerpt)
```

```
<%= Title %>
```

If you're familiar with classic ASP, you'll know what this code does: it simply outputs the value of the variable Title that we declared in the previous inline code block.

❖ **WebForms:**

Above Fig2.2 code shows form tage having attribute runat=”Server” is called Web Form. Anything contained in a web form can be handle by asp.net

❖ **Web Server Controls**

Above Fig.2.2 shows some of the label and textbox controls which are having attributes runat=”sever” knows as Web Controls.