STUDENT ACTIVITY 1.2

MTA Course: Web Development Fundamentals
Topic: Understand ASP.NET intrinsic objects
File name: WebDevFund_SA_1.2

Lesson Objective:

1.2: Understand ASP.NET intrinsic objects. *This objective may include but is not limited to:* Request, Server, Application, Session, Response, HttpContext

Resources, software, and additional files needed for this lesson:

- Microsoft® Visual Basic 2008®, Express Edition
- Microsoft Visual C# 2008®, Express Edition
- Microsoft Visual Web Developer 2008, Express Edition
- WebDevFund_SA_1.2
- Use this site for additional information: *http://msdn.microsoft.com/en-us/rampup/dd861531.aspx*

Directions to the student:

Review the following example from the Ramp-Up site. Use this code to create a Web page that uses the ASP.NET controls discussed in this lesson. Both Visual Basic® and C# versions are included.

Example ASP.NET Web Page

The following code example shows a page that includes the basic elements that constitute an ASP.NET Web page. The page contains static text as you might have in a Hypertext Markup Language (HTML) page, along with elements that are specific to ASP.NET.

Visual Basic code

```
<%@ Page Language="VB" %>
<html>
<head runat="server">
  <title>Basic ASP.NET Web Page</title>
<script runat="server">
    Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        Label1.Text = "Welcome, " & TextBox1.Text
    End Sub
</script>
</head>
<body>
  <form id="form1" runat="server">
    <h1>Welcome to ASP.NET</h1>
    <p>Type your name and click the button.</p>
    <p>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server"
        Text="Click" OnClick="Button1_Click" />
    </p>
    <p>
      <asp:Label ID="Label1" runat="server"></asp:Label>
    </p>
  </form>
</body>
</html>
```

C# code

```
<%@ Page Language="C#" %>
<html>
<head runat="server">
  <title>Basic ASP.NET Web Page</title>
<script runat="server">
    void Button1_Click(Object sender, EventArgs e){
        Label1.Text = "Welcome, " + TextBox1.Text;
```

```
    }
</script>
</head>
<body>
  <form id="form1" runat="server">
    <h1>Welcome to ASP.NET</h1>
    <p>Type your name and click the button.</p>
    <p>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server"
        Text="Click" OnClick="Button1_Click" />
    </p>
    <p>
      <asp:Label ID="Label1" runat="server"></asp:Label>
    </p>
  </form>
</body>
</html>
```

Form Elements

If your page includes controls that allow users to interact with the page and submit it, the page must include a **form** element. You use the standard HTML **form** element, but certain rules apply. The rules for using the **form** element are as follows:

- The page can contain only one **form** element.
- The **form** element must contain the **runat** attribute with the value set to **server**. This attribute allows you to refer to the form and the controls on the page programmatically in server code.
- Server controls that can perform a postback must be inside the **form** element.
- The opening tag must not contain an **action** attribute. ASP.NET sets these attributes dynamically when the page is processed, overriding any settings that you might make.

Web Server Controls

In most ASP.NET pages, you will add controls that allow the user to interact with the page, including buttons, text boxes, lists, and other elements. These Web server controls are similar to HTML buttons and **input** elements. However, they are processed on the server, allowing you to use server code to set their properties. These controls also raise events that you can handle in server code.

Server controls use a special syntax that ASP.NET recognizes when the page runs. The following code example shows some typical Web server controls:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server"  Text="Click" OnClick="Button1_Click"
/>
```

The tag name for ASP.NET server controls starts with a prefix— in this case, `asp:`. The prefix might be different if the control is not part of the Microsoft .NET Framework. ASP.NET server controls also include the **runat="server"** attribute and, optionally, an ID that you can use to reference the control in server code.

When the page runs, it identifies the server controls and runs the code that is associated with those controls. Many controls render some HTML or other markup into the page. For example, the **asp:textbox** control renders an **input** element with the **type="text"** attribute into a page. However, there is not necessarily a one-to-one mapping between a Web server control and an HTML element. For example, the **asp:calendar** control renders an HTML table. Some controls do not render anything to the browser; instead, they are processed only on the server, and they provide information to other controls.

HTML Elements as Server Controls

Instead of, or in addition to, using ASP.NET server controls, you can use ordinary HTML elements as server controls. You can add the **runat="server"** attribute and an **ID** attribute to any HTML element in the page. When the page runs, ASP.NET identifies the element as a server control and makes it available to server code. For example, you can add the required elements to an HTML **body** element, as shown in the following code example:

```
<body runat="server" id="body">
```

You can then reference the **body** element in server code—for example, to set the body background color at run time in response to user input or to information from a database.

Server Code

Most ASP.NET pages include code that runs on the server when the page is processed. ASP.NET supports many languages, including C#, Visual Basic, J#, Jscript®, and others.

ASP.NET supports two models for writing server code for a Web page. In the single-file model, the code for the page is in a **script** element where the opening tag includes the **runat="server"** attribute. The example earlier in this topic shows the single-file model.

Alternatively, you can create the code for the page in a separate class file, which is referred to as the *code-behind model.* In this case, the ASP.NET Web page generally contains no server code. Instead, the **@ Page** directive includes information that links the .aspx page with its associated code-behind file. The following code example shows a typical **@ Page** directive for a page with a code-behind file.

Visual Basic code

```
<%@ Page Language="VB" CodeFile="Default.aspx.vb" Inherits="Default" %>
```

C# code

```
<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="Default" %>
```

The **CodeFile** attribute specifies the name of the separate class file, and the **Inherits** attribute specifies the name of the class within the code-behind file that corresponds to the page.

Embedded Code Blocks in ASP.NET Web Pages

The default model for adding code to an ASP.NET Web page is to either create a code-behind class file (a code-behind page) or to write the page's code in a **script** block with the attribute **runat="server"** (a single-file page). The code you write typically interacts with controls on the page. For example, you can display information on the page from code by setting the **Text** (or other) properties of controls.

Another possibility is to embed code directly into the page using an embedded code block.

Embedded Code Blocks

An embedded code block is server code that executes during the page's render phase. The code in the block can execute programming statements and call functions in the current page class.

The following code example shows an ASP.NET page with an embedded code block that displays the results of a loop.

Visual Basic code

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

<html><head><title></title></head>

<body>

    <form id="form1" runat="server">

    <% For i As Integer = 0 To 5 %>

       <% Response.Write("<br>" & i.ToString())%>

    <% Next%>

    </form>

</body>

</html>
```

C# code

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

```
<html>
<head><title></title></head>
<body>
    <form id="form1" runat="server">
    <% for(int i = 0; i < 6; i++) %>
       <% { Response.Write("<br>" + i.ToString()); }%>
    </form>
</body>
</html>
```

The following code example shows an embedded code block that displays the value of a public `GetTime()` function inside a span element. In embedded code blocks, the syntax `<% = expression %>` is used to resolve an expression and return its value into the block.

Visual Basic code

```
<%@ Page Language="VB" %>
<html>
<head>
<script runat=server>
Protected Function GetTime() As String
    Return DateTime.Now.ToString("t")
End Function
</script>
</head>
<body>
    <form id="form1" runat="server">
       Current server time is <b><% =GetTime()%></b>.
    </form>
</body>
</html>
```

C# code

```
<%@ Page Language="C#" %>
<html>
<head>
<script runat=server>
protected String GetTime()
```

```
{
    return DateTime.Now.ToString("t");
}
</script>
</head>
<body>
    <form id="form1" runat="server">
       Current server time is <b><% =GetTime()%></b>.
    </form>
</body>
</html>
```

Embedded code blocks must be written in the page's default language. For example, if the page's **@ Page** directive contains the attribute `language="VB",` the page will use the Visual Basic compiler to compile code in any script block marked with `runat="server"` and any inline code in `<% %>` delimiters

Uses for Embedded Code Blocks

Embedded code blocks are supported in ASP.NET Web pages primarily to preserve backward compatibility with older ASP technology. In general, using embedded code blocks for complex programming logic is not a best practice because when the code is mixed on the page with markup, it can be difficult to debug and maintain. In addition, because the code is executed only during the page's render phase, you have substantially less flexibility than with code-behind or script-block code in scoping your code to the appropriate stage of page processing.

Some uses for embedded code blocks include:

- Setting the value of a control or markup element to a value returned by a function, as illustrated in the preceding example.
- Embedding a calculation directly into the markup or control property.