msdn® training

Module 1: Working with Microsoft ASP.NET

Contents

Microsoft

Overview

- Introducing ASP.NET
- Creating Web Forms
- Adding ASP.NET Code to a Page
- Handling Page Events
- Discussion: ASP vs. ASP.NET

Microsoft® Active Server Pages (ASP) technology is widely used to create dynamic Web sites and applications. However, ASP has several limitations, such as the need for redundant and lengthy coding to accomplish simple goals. To overcome these limitations of ASP, Microsoft has developed a new technology called Microsoft ASP.NET, which is a part of the Microsoft .NET strategy for Web development. ASP.NET is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications.

In this module, you will learn about the main features of ASP.NET and discover the differences between ASP and ASP.NET. You will also learn about server controls and see how to add server-side script to an ASP.NET page.

After completing this module, you will be able to:

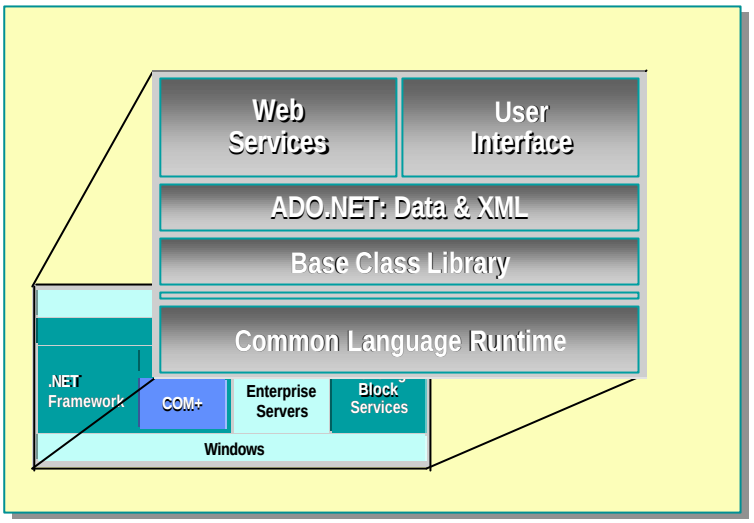
- Identify the main features of ASP.NET.
- Identify the differences between ASP and ASP.NET.
- Describe the working model of ASP.NET.
- Describe the architecture of server controls.
- Add a Hypertext Markup Language (HTML) server control to a page.
- Access the properties and methods of server controls in code.
- Add event handlers for page events.
- Use the **IsPostback** property to handle postback forms.

Introducing ASP.NET

- The Microsoft .NET Framework
- ASP.NET Features
- Animation: ASP.NET Execution Model

In this section, you will read an overview of the Microsoft .NET Framework and see how ASP.NET fits in. Next, you will learn about the various features of ASP.NET and see a working model. You will also learn about the main differences between ASP and AS P.NET.

The Microsoft .NET Framework



The Microsoft .NET Platform provides all of the tools and technologies that are needed to build distributed Web applications. It exposes a consistent, language-independent programming model across all tiers of an application, while providing seamless interoperability with and easy migration from existing technologies.

The Microsoft .NET Platform is composed of several core technologies:

- The Microsoft .NET Framework
- The Microsoft .NET Building Block Services
- The Microsoft .NET Enterprise Servers
- Microsoft Visual Studio® .NET
- Microsoft Windows. NET

The Microsoft .NET Framework is a set of technologies that is integral to the Microsoft .NET Platform. It provides the basic building blocks for developing Web applications and services. The Microsoft .NET Framework includes the following components:

- Common Language Runtime
- Base class library
- Data
- Web forms and Web services
- WinForms

Common Language Runtime

The Common Language Runtime provides the programming interface between the Microsoft .NET Framework and the programming languages available for the Microsoft .NET Platform. It simplifies application development, provides a robust and secure execution environment, supports multiple languages, and simplifies application deployment and management. The run time loads and runs code written in any run time-aware programming language. Code that targets the run time is called managed code. Managed code simply means that there is a defined contract of cooperation between natively executing code and the run time itself. Responsibility for tasks like creating objects, making method calls, and so on is delegated to the run time, which enables the run time to provide additional services to the executing code.

Base Classes and Libraries

The Microsoft .NET Framework includes classes that encapsulate data structures, perform Input/Output (I/O), provide access to information about a loaded class, and provide a way to invoke security checks. It also includes classes that encapsulate exceptions and other helpful functionality such as data access, server-side user interface (UI) projections, and rich graphical user interface (GUI) generation. The Microsoft .NET Framework provides both abstract base classes and class implementations derived from those base classes. You can use these derived classes "as is" or derive your own classes from them.

The Microsoft .NET Framework classes are named using a dot-syntax naming scheme that connotes a naming hierarchy. This technique is used to group related classes logically together so that they can be searched and referenced more easily. A grouping of classes is called a namespace. For example, a program can use classes in the System.Data.SqlClient namespace to read data from a SQL Server database.

The root namespace for the Microsoft .NET Framework is the **System** namespace.

Data

Microsoft ADO.NET is the next generation of ActiveX® Data Object (ADO) technology. ADO.NET provides improved support for the disconnected programming model and also provides rich Extensible Markup Language (XML) support. ADO was created to provide data services to traditional client applications that were tightly coupled to the database; consequently it was not effective for Web applications. ADO.NET was created with the characteristics of Web applications in mind.

Web Forms and Web Services

ASP.NET is a programming framework built on the Common Language Runtime that can be used on a server to build powerful Web applications. ASP.NET Web forms provide an easy and powerful way to build dynamic user interfaces. ASP.NET Web services provide the building blocks for constructing distributed Web-based applications. Web services are based on the Simple Object Access Protocol (SOAP) specification.

Win Forms

For applications that are based on Microsoft Windows®, the Microsoft .NET Framework provides the **System.Windows.Forms** namespace to create the user interface. You can use **System.Windows.Forms** to do rapid application design (RAD). It provides inheritance in the same client user interface library. You can build components by using inheritance and then aggregate them by using a form designer such as Microsoft Visual Basic®.

ASP.NET Features

- **Multiple Language Support**
- **Increased Performance**
  - Compiled code
  - Cache
- **Classes and Namespaces**
- **Server Controls**
- **Web Services**

ASP.NET is more than just the next version of ASP. It is a totally re-architected technology for creating dynamic, Web-based applications. ASP pages use the **.**asp extension and ASP.NET pages use the extension .aspx.

Note Both ASP and ASP.NET pages can be used on the same Web site. Existing ASP pages will still work along with new ASP.NET pages; they do not need to be converted into ASP.NET pages.

ASP.NET, with a host of new features, allows developers to write cleaner code that is easy to reuse and share. ASP.NET boosts performance and scalability by offering access to compiled languages. Some of the main features of ASP.NET are described below.

Multiple Language Support

ASP.NET provides a true language-neutral execution framework for Web applications.

You can currently use over 20 languages to build .NET applications. Microsoft has compilers for Visual Basic, Microsoft Visual C#™, Microsoft Visual C++®, and Microsoft JScript®. Third-party vendors are writing .NET compilers for Cobol, Pascal, Perl, and Smalltalk, among others.

The labs and the sample code in this course will use Visual Basic.

Increased Performance

In ASP.NET, code is compiled. When you request a page for the first time, the run time compiles the code and the page itself, and keeps a cached copy of the compiled result. When you request the page the second time, the cached copy is used. This results in greatly increased performance because, after this first request, the code can run from the much faster compiled version and the content on the page does not need to be parsed again.

Classes and Namespaces

ASP.NET includes a range of useful classes and namespaces. Namespaces are used as an organizational system—a way to present program components that are exposed to other programs and applications. Namespaces contains classes.

Namespaces are like class libraries and can make writing Web applications easier. Some of the classes included with ASP.NET are **HtmlAnchor**, **HtmlControl**, and **HtmlForm**, which are included within the **System.Web.UI.HtmlControls** namespace.

Note Namespaces may change between the Beta 2 and the final release versions of ASP.NET.

Server Controls

ASP.NET provides several server controls that simplify the task of creating pages. These server controls encapsulate common tasks that range from displaying calendars and tables to validating user input. They automatically maintain their selection states and expose properties, methods, and events for server-side code, thereby providing a clean programming model.

For more information about using server controls, see Module 2, "Using Web Controls," in Course 2063B, *Introduction to MicrosoftASP.NET*.

Web Services

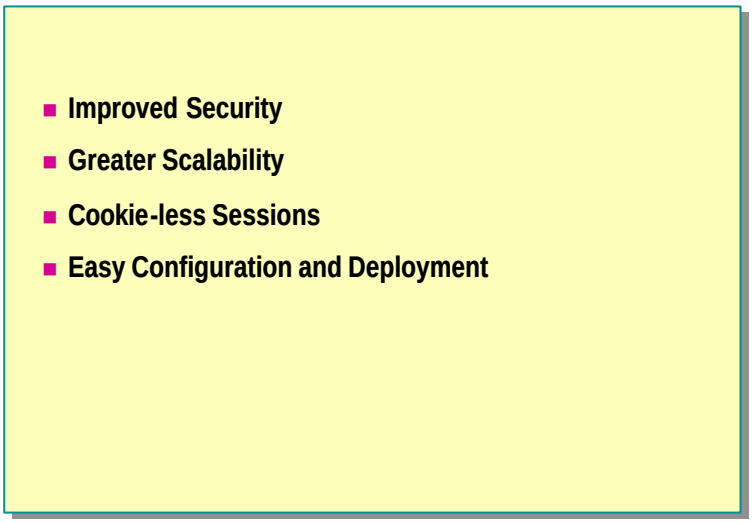
A Web service is an application delivered as a service that can be integrated with other Web services by using Internet standards. ASP.NET allows you to use and create Web services.

For example, a company can assemble an online store by using the Microsoft Passport service to authenticate users, a third-party personalization service to adapt Web pages to each user's preferences, a credit-card processing service, a sales tax service, package-tracking services from each shipping company, and an in-house catalog service that connects to the company's internal inventory management applications.

Web services provide the building blocks for constructing distributed Web-based applications. ASP.NET files have an .aspx extension and Web services have an .asmx extension. The technologies are similar; however, instead of outputting HTML, a Web service outputs a computer-readable answer to the input it receives.

For more information about Web services, see Module 6, " Using Web Services," in Course 2063B, *Introduction to Microsoft ASP.NET*.

ASP.NET Features (*continued*)



Improved Security

In ASP, the only type of authentication that you can use is the Windows authentication, whereas ASP.NET allows different types of logon and user authentication: Windows, Passport, and Cookies.

ASP.NET also enables you to get real account impersonation and have the server execute code as if the user were present. You can also programmatically check to see if the user is in a given role and conditionally let him or her perform certain tasks when given permission.

In addition, creating forms-based authentication, in which you can create your own custom logon screen and credential checking, is much easier if you are using ASP.NET.

For more information about authentication and creating login forms, see Module 7, "Creating a Microsoft ASP.NET Web Application," in Course 2063B, *Introduction to Microsoft ASP.NET.*

Greater Scalability

In ASP.NET, session state can now be maintained in a separate process on a separate machine or database, allowing for cross-server sessions. This allows you to add more Web servers as your traffic grows.

Cookie-less Sessions

ASP.NET enables you to use session state even with browsers that have cookie support disabled. Cookie-less sessions use Uniform Resource Locators (URLs), as opposed to cookies, to pass the SessionID to an ASP.NET page. A cookie-less session involves encoding data into a URL, which is done automatically by the browser.

For more information about maintaining state, see Module 7, "Creating a Microsoft ASP.NET Web Application," in Course 2063B, *Introduction to Microsoft ASP.NET*.

Easy Configuration and Deployment

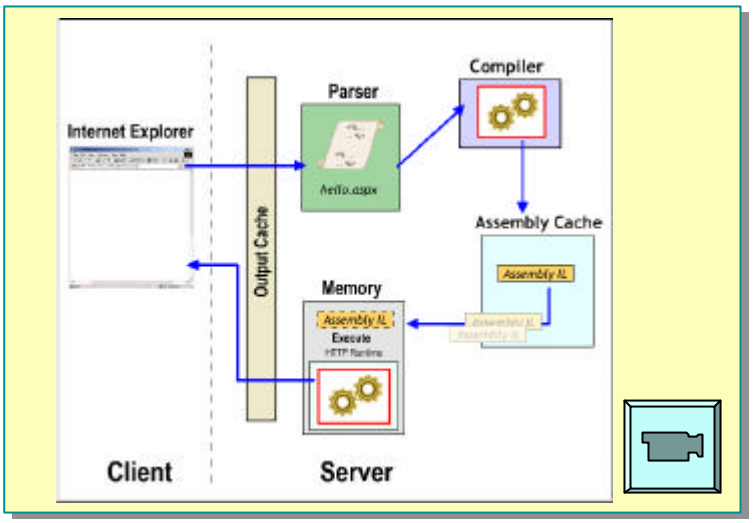
Configuration and deployment are now easier with the use of human-readable configuration files and DLLs that do not need to be registered.

In ASP.NET, all the configuration details for all Web applications are kept in human-readable files named web.config. The standard format for the configuration files is XML, and each application inherits the settings in the default web.config file.

With .NET, all files that a Web site needs are located under the site's root folder. DLLs are in the /bin folder, and the web.config file is in the root folder. To deploy a Web site, all you need to do is copy the site's root folder by using file copy commands, the Microsoft FrontPage® server extensions, or File Transfer Protocol (FTP).

For more information about configuration options, see Module 7, "Creating a Microsoft ASP.NET Web Application," in Course 2063B, *Introduction to Microsoft ASP.NET*.

Animation: ASP.NET Execution Model



In this animation, you will see how ASP.NET pages are processed on the server. To view the animation, open the **2063B_01A001.swf** file from the **Media** folder.

The following table describes the elements that are depicted in the animation.

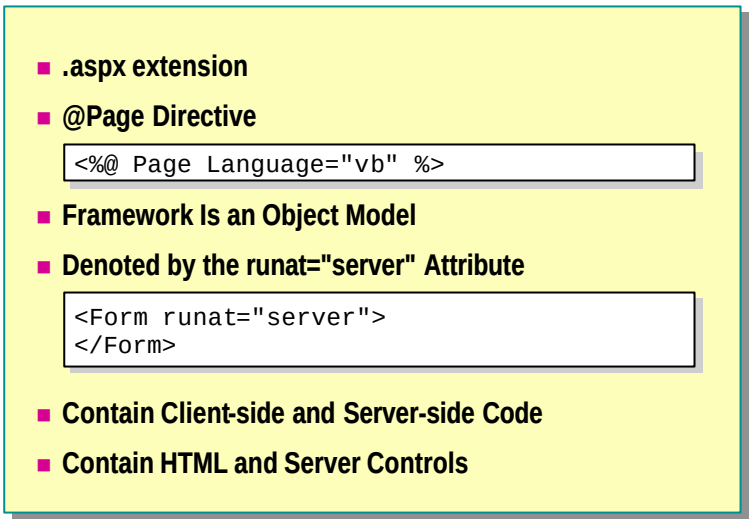
Element	Description
Parser	The parser checks and interprets the contents of the aspx page and passes the page to a compiler.
Compiler	The run-time compiler is responsible for compiling the contents of the page into an intermediate language.
Assembly Cache	Each machine on which the Microsoft .NET Framework is installed has a machine-wide code cache called the assembly cache. One of the functions of the assembly cache is to hold the native code versions of pages that have been pre-compiled.
Memory	Some items that are expensive to construct can be built once and used for a period of time before they are considered invalid. These items are stored in memory where they can be efficiently retrieved without incurring the cost of being reconstructed.
Output Cache	The output cache is a cache for entire pages, including their objects and data. After a page is built, it can be placed in the output cache. If a user makes another request for the page, the request is returned from the output cache.

◆ Creating Web Forms

- What Are Web Forms?
- What Are Server Controls?
- Types of Server Controls
- How Do Server Controls Work?
- Demonstration:
  Adding Server Controls to an ASP.NET Page

ASP.NET introduces the concept of server controls that encapsulate common tasks and provide a clean programming model. In this section, you will learn about types of server controls and see how they work. You will also learn how to use server controls on ASP.NET pages.

What Are Web Forms?



Web Forms divide Web applications into two pieces: the visual component and the user interface logic.

- Web Forms have an .aspx extension

  Web forms are commonly referred to as ASP.NET pages or ASPX pages. They have an .aspx extension and work as the containers for the text and controls that you want to display.

- @Page directive

  The @Page directive defines page-specific attributes that are used by the ASP.NET page parser and compiler. You can set the language attribute to the language that will be used throughout all code on the page.

  You can include only one @ Page directive per .aspx file.

- The Web Forms framework is an object model

  Although you create Web forms from separate components, they form a unit. When the Web form is compiled, ASP.NET parses the page and its code, generates a new class dynamically, and then compiles the new class. The dynamically generated class is derived from the ASP.NET Page class, but is extended with controls, your code, and the static HTML text in the .aspx file. This is different from ASP. In ASP, the page consists of static HTML interspersed with executable code. The ASP processor reads the page, extracts and runs the code, and then inserts the results back into the static HTML before sending the results to the browser.

  Unlike controls in an ASP form, all intrinsic controls in an ASP.NET form are objects. Therefore, all the controls on a form have properties, methods, and events.

- A Web form is denoted by the runat="server" attribute

  The runat="server" attribute ensures that the form is executed at the server.

  ```
  <Form runat="server">

  </Form>
  ```

- Contain client-side and server-side code

  In the previous versions of ASP, controls on forms could invoke only client-side functions. This has been changed in ASP.NET by the introduction of server controls and server-side events.

  Like ASP pages, Web forms can contain client-side and server-side code.

- Contain HTML and server controls

  Server controls are new in ASP.NET. Web forms can contain HTML and server controls.

Note ASP.NET and ASP pages can coexist on the same machine. The file extension determines only whether ASP or ASP.NET processes it.

What Are Server Controls?

- Server-programmable Objects
- Denoted by Tag with the runat = "server" Attribute
- Encapsulate Both Behavior and Rendering
- Fully Declarative
- Render Different HTML to Support Multiple Browsers or Other Web Clients

ASP allowed developers to execute components on the server. ASP.NET extends this concept with the introduction of server controls.

Server controls are controls that have built-in behavior. They have properties, methods, and events that can be accessed at run time from code running on the server.

They provide client-specific HTML that is displayed on the client. This means that you do not need to create separate pages for each browser type, nor query what type of browser is being used, because the control does that for you. For example, server controls can determine whether a presentation task, such as making a Web page text dynamically appear or disappear, should be performed by the browser or server code. If Microsoft Internet Explorer, version 4.0 or above, was the destination, the code is on the client side because Internet Explorer 4.0 can handle dynamic text operations. If an older version of Internet Explorer (for example, Internet Explorer 3.0) is the destination, the code is on the server side, resulting in an extra transaction with the server to change the text. From the user point of view, the text would change, but depending on the browser type, changes would occur either on the client or server. All this can be accomplished by a server control, which determines the browser type and generates the appropriate code.

Server controls also provide a consistent object model for the controls, providing standard properties, methods, and events.

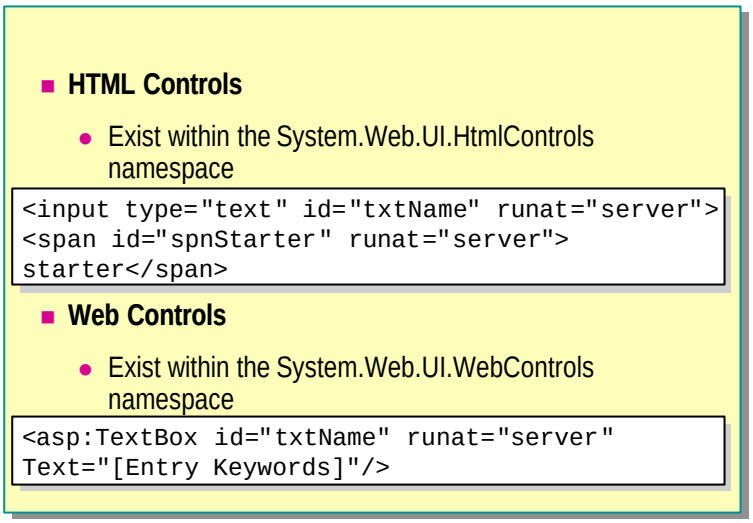
Users can easily create server controls from existing HTML controls. To create a server control, simply add the **runat** attribute to the control's tag. For example, the following is an HTML input box, turned into a server control:

```
<INPUT TYPE="TEXT" runat="server">
```

Note Without the runat="server" attribute, this line of HTML would be parsed into a standard HTML text box.

After you create a server control, you can access its properties, methods, and events through server-side code, making it easy to obtain user input and provide user feedback.

Types of Server Controls



Server controls are used to create the user interface for your Web application. They can generate any output that is suitable for the device or browser they are targeting and can maintain state between trips to the server. There are two sets of server controls in the ASP.NET Framework: HTML controls and Web controls. Both render HTML that is displayed by Web browsers.

HTML Controls

HTML controls offer Web developers the power of the Web Forms page framework while retaining the familiarity and ease of use of HTML elements. These controls look exactly like HTML, except that they have a **runat="server"** attribute/value pair in the opening tag of the HTML element. For example, the following HTML not only creates a text box on a Web page, but also creates an instance of the text box server control:

```
<input type="text" runat="server" id="txtName" value="some
text">
```

To enable programmatic referencing of the control, include a unique **id** attribute. In the preceding example, the id="txtName" defines this programmatic **id**, allowing developers to manipulate the contents of this text box with server-side events and other code.

ASP.NET offers direct object model support for the most commonly used HTML elements. For object models that are not directly supported, there is the **HtmlGenericControl** object, which supports the , , <body>, and <font> elements, among others.

For example, you can use the following code to create a SPAN server element.

```
<span id="spnStarter" runat="server">starter text</span>
```

The HTML controls exist in the System.Web.UI.HtmlControls namespace.

Note All HTML controls must be well-formed and must not overlap. Unless otherwise noted, elements must be closed, either with an ending slash within the tag or with a closing tag, like the XML syntax.

Web Controls

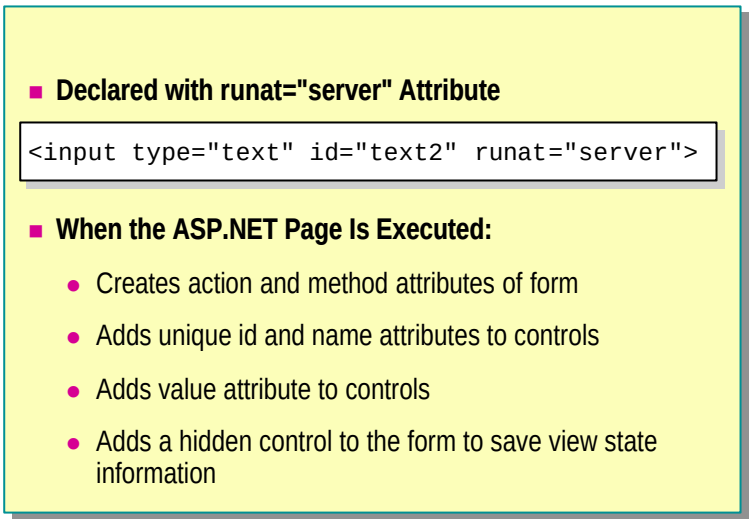
Web controls include traditional form controls such as the **TextBox** and **Button** controls, as well as other higher-level abstractions such as the **Calendar** and **DataGrid** controls. Web controls can be further classified into **Intrinsic**, **Rich**, **Validation**, and **List** controls.

Web controls appear in the HTML markup as namespaced tags—that is, tags with a prefix. The prefix is used to map the tag to the namespace of the run-time component. The remainder of the tag is the name of the run-time class itself. Like HTML controls, these tags must also contain a **runat="server"** attribute. An example declaration is as follows:

```
<asp:TextBox id="txtName" runat="server" Text="[Entry
Keywords]"></asp:TextBox>
```

In the preceding example, "asp" is the namespace tag prefix and it maps to the **System.Web.UI.WebControls** namespace. This namespace is automatically included in an ASP.NET page; you do not need to import it.

How Do Server Controls Work?



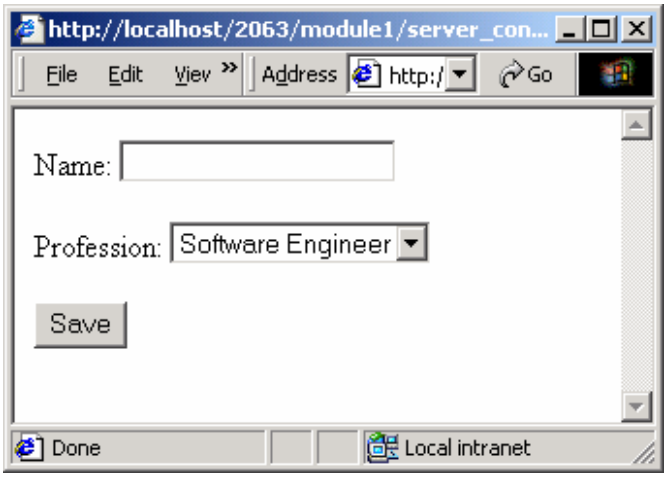
The key to the working of server controls is the **runat="server"** attribute. When you set the **runat** attribute, you enable the server-side events and view state maintenance for controls. If you do not set the **runat** attribute, the control works as a plain HTML control. The state for the controls is not maintained and no server-side events are available. You can have both server and plain HTML controls on a form.

Web controls usually reside on a form. Forms also have the **runat="server"** attribute. The **runat** attribute enables forms to maintain the view state of their controls in an ASP.NET page. When an ASP.NET page is submitted to the server, ASP.NET automatically adds a hidden control named **_VIEWSTATE** to the form. If the state of a control has been modified, the **_VIEWSTATE** control is used to remember the values. This way, changes made to a page can be saved across multiple requests.

To understand how server controls work, consider an example.

Example

The following example is a simple form that allows you to type your name and select your job title from a list box.




```
<FORM runat="server">
Name: <INPUT type="text" runat="server">
<p>Profession: <SELECT runat="server">
      <OPTION> Software Engineer </OPTION>
      <OPTION> Software Tester </OPTION>
      <OPTION> Program Manager </OPTION>
</SELECT>
<p><INPUT type="Submit" Value="Save" runat="server">
</FORM>
```

Assume that a software engineer named Jane fills in and submits this form.

When your ASP.NET code is executed, it does the following:

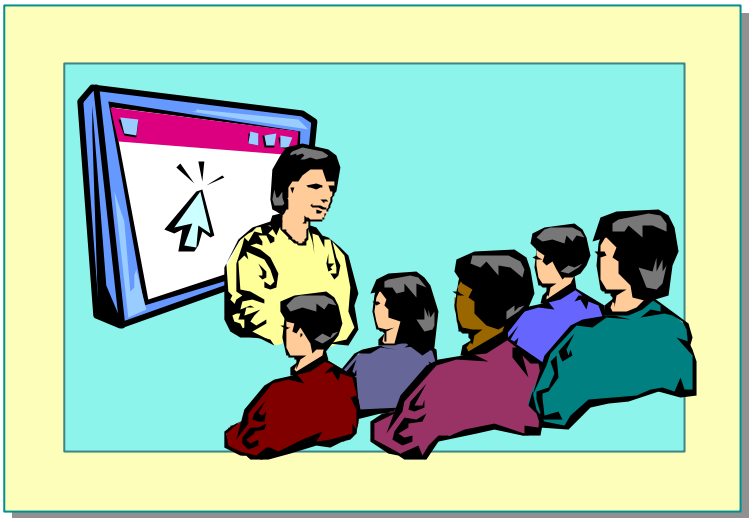
1. Creates the **action** and **method** attributes of the form so that the values of the controls in the form are posted back to the same page.
2. Adds a unique id and **name** attribute to the form. However, if you specify these on the form tag, the values that you specify will be used.
3. For each control, adds the **value** attribute with the text value that was in the control when the form was submitted. This maintains the state of the server controls.
4. Adds a hidden control named **__VIEWSTATE** to the form that stores state changes of a page.

This is the HTML that is returned to the browser:

```
<FORM name="ctrl1" method="post" action="server_controls.aspx"
id="ctrl1">
<INPUT type="hidden" name="__VIEWSTATE"
value="a0z-1466413384__x">

  Name: <INPUT value="Jane" name="ctrl3" type="text">
  <p>
  Profession:   <SELECT name="ctrl5">
      <OPTION value="Software Engineer">
          Software Engineer</OPTION>
      <OPTION value="Software Tester">
          Software Tester</OPTION>
      <OPTION selected value="Program Manager">
          Program Manager</OPTION>
  </SELECT>
  <p>
  <input type="submit" value="Save">
</FORM>
```

Demonstration: Adding Server Controls to an ASP.NET Page



In this demonstration, you will see how to add HTML server controls to an ASP.NET page.

↙ To run this demonstration

1. Open the file <*install folder*>\Democode\Mod01\server_controls.aspx with Microsoft Notepad.

   This file has a form with an input box, a list box, and a **Submit** button.
2. View the page in Internet Explorer.
3. Enter information in the controls and click **Save**.

   When the page is re-displayed, the information in the controls is lost.
4. Edit the page and add a **runat="server"** attribute to the three controls.
5. View the page in the browser again. View the source of the page to show the changes that were made.

   Among other changes made, a **name** attribute was added to each control.
6. Enter information in the controls and click **Save**. The controls still lose their values.
7. Edit the page and add a **runat="server"** attribute to the form.
8. View the page in the browser again. View the source of the page to show the changes that were made.

   Among other changes made, **action** and **method** attributes were added to the form and a hidden control was created.
9. Enter information in the controls and click **Save**. The controls now save their values.

10. Edit the page and add an intrinsic label Web control to the page:

```
<asp:label id="label1" runat="server"/>
```

11. View the page in the browser and show the source of the page.

    The intrinsic control generates a element.

↙ To demonstrate how to use Microsoft Visual Studio® .NET

1. Open Visual Studio .NET and create a new Visual Basic ASP.NET Web Application project.

   A number of files are added to the project by default. For most of the course, we will be working only with ASPX files.

   An ASPX file, WebForm1.aspx, is created by default and opened in Design view.
2. In the Solution Explorer window, click the **Show All Files** button.
3. By default, a number of windows are opened in Visual Studio .NET. Windows can be "pinned" to the work area, or open only when you want to see them. Click the **Auto Hide** button in the upper right corner of the Solution Explorer window. Now the window will open only when you hover over or click its tab on the side of the workspace.
4. Click the Toolbox tab to display the toolbox window.
5. Add a text box and a button to the default Web form by dragging and dropping the controls from the toolbox onto the page.

   The **Properties** window is a great way to learn about the properties of the Web controls.
6. A code-behind page is created by default for the ASPX page named WebForm1.aspx.vb. Double-click the button to create an OnClick event procedure in the code-behind page.

   Note You will not be using this feature of Visual Studio .NET in this course. You will create event procedures manually.

---

7. Go back to the WebForm1.aspx page and click the **HTML** tab to go into HTML view.

   By default, ASPX pages are created with code-behind pages. To remove this feature, delete all the attributes in the **@Page** directive except the **Language** attribute.
8. Add an HTML text box to the default form. Microsoft IntelliSense® is built into Visual Studio .NET and supplies many of the attributes of the control.

```
<input type="text" runat="server">
```

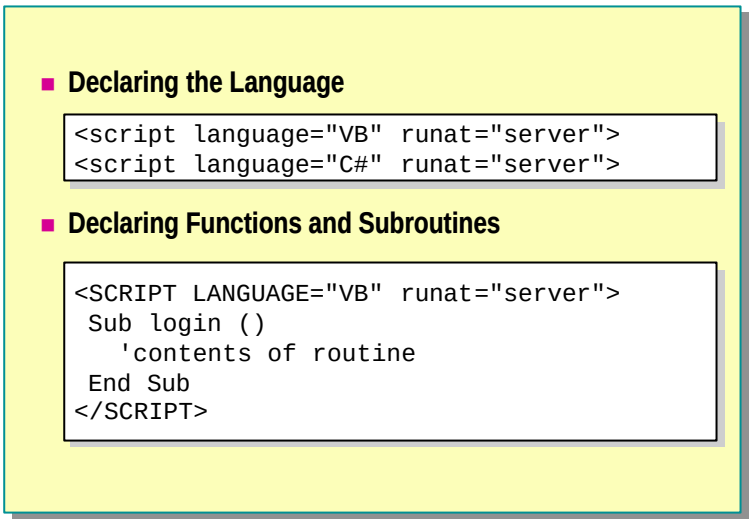
9. Add a <script> section to the page by right-clicking on the page, clicking **Insert Script Block**, and then clicking **Server**.
10. To view the page, right-click and click **View in Browser**.

Adding ASP.NET Code to a Page

- Creating an ASP.NET <SCRIPT> Section
- Visual Basic 6.0 vs. Visual Basic. NET Version 7.0
- Creating Event Procedures
- Demonstration: Adding Code to Controls

ASP.NET introduces a new way of coding that is very similar to coding in event-driven languages, such as Visual Basic and dynamic HTML (DHTML) scripting. In this section, you will learn how to create a basic ASP.NET page. You will learn about the major differences between Visual Basic version 6.0 and Visual Basic .NET version 7.0. Finally, you will also learn how to add functionality to the controls on an ASP.NET page.

Creating an ASP.NET <SCRIPT> Section



Most code in an ASP.NET page is placed in <SCRIPT> sections. You can use the ASP <% and %> syntax, but it is not the preferred method. Although <% %> code blocks provide a powerful way to custom-manipulate the text output returned from an ASP.NET page, they do not provide much help in providing a clean HTML programming model. Developers using only <% %> code blocks must custom-manage page state between round trips and custom interpret posted values.

When you create a <SCRIPT> section for server-side code, you need to declare the language being used for the code and set the **runat** attribute to "server". The default language is Visual Basic, which is a superset of Microsoft Visual Basic Scripting Edition (VBScript). For example, the following code example declares Visual Basic as the language.

```
<SCRIPT LANGUAGE="VB" runat="server">
```

In ASP.NET, you declare functions and sub-procedures within the <SCRIPT> tags.

```
<SCRIPT LANGUAGE="VB" runat="server">
  Sub login()
      'contents of routine
  End Sub
</SCRIPT>
```

Visual Basic 6.0 vs. Visual Basic .NET Version 7.0

- No More Set and Let
- Need to Use Parentheses When Calling Functions and Sub-Procedures
- Parameters Are Passed by Value
- You Can Specify the Data Type of a Variable; Variants Are Not Supported
- Variables Can Be Initialized When They Are Declared
- Improved Type Safety
- Structured Error Handling with Try/Catch/Finally
- New Conversion Methods

Visual Basic .NET version 7.0 is very different from earlier versions. Following are some of the major differences between Visual Basic .NET version 7.0 and earlier versions of Visual Basic.

- No more **Set** and **Let**

  You assign values to object variables; you no longer use the **Set** syntax. This is because the concept of default properties and methods does not exist any more. Furthermore, when comparing objects, you must use the **is** operator, not the = operator:

  ```
  objThis is objThat
  ```

- Use parentheses when calling functions and sub-procedures

  All methods, functions, and sub-procedures must now use parentheses to enclose the parameter list.

  ```
  Response.Write ("Hello")
  ```

- Parameters are passed by value

  Previously, parameters were passed by reference in Visual Basic. Now they are passed by value. You can add the **ByRef** keyword to the parameter list if you want to pass a parameter by reference.

- Declaring variables

  As in Visual Basic 6.0, you can specify the data type of a variable when you declare it. However, the **Variant** data type is no longer supported in Visual Basic .NET.

  ```
  Dim strName as String
  Dim strTitle, strAddress as String
  ```

  You can turn on Option Explicit by setting the **Explicit** attribute of the @Page directive to true.

  ```
  <%@Page Language="vb" Explicit="true" %>
  ```

- Initializers

  Visual Basic .NET version 7.0 supports initialization of variables on the line in which they are declared. Initializers can be used anywhere, including inside a control structure. The semantics of a procedure-level declaration, which include an initalizer, are the same as those for a declaration statement immediately followed by an assignment statement. In other words, the following statement:

  ```
  Dim X As Integer = 1
  ```

  is equivalent to these statements:

  ```
  Dim X As Integer
  X = 1
  ```

- Type Safety

  Visual Basic .NET version 7.0 offers improved type safety by generating errors when a conversion that could fail at run time or is unexpected by the user is required.

- Structured Error Handling

  In the past, developers needed to provide error-handling code in every function and subroutine. Developers have found that a consistent error-handling scheme means writing a great deal of duplicated code.

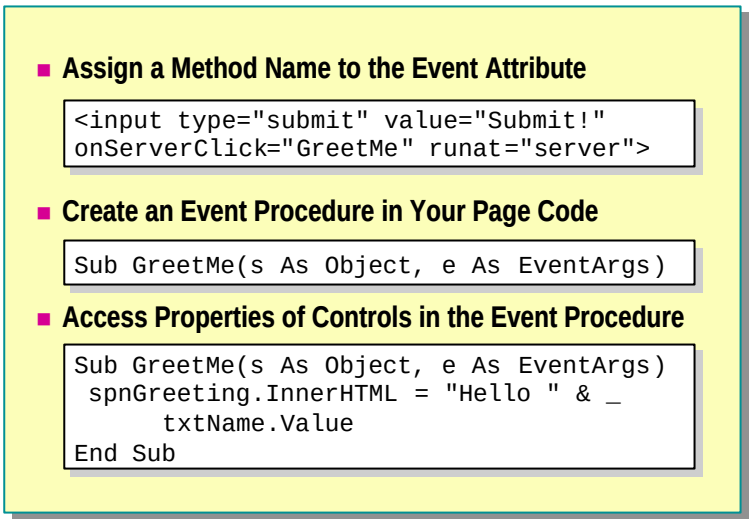
  With **Try...Catch...Finally**, these problems are eliminated. Developers can nest their exception handling, and there is a control structure for writing cleanup code that executes in both normal and exception conditions.

  ```
  Sub SEH()
     Try
         Open "TESTFILE" For Output As #1
         Write #1, CustomerInformation
     Catch
         Kill "TESTFILE"
     Finally
         Close #1
     End try
  End Sub
  ```

- New Conversion Methods

  Visual Basic .NET supports new conversion methods such as **ToString** for converting data types. You can still use conversion functions such as **CStr**, but **ToString** is the recommended method.

Creating Event Procedures



You can create event handlers for server controls by using the ASP.NET syntax. In the event handlers, you can access the properties of a control.

↙ To create event handlers by using the ASP.NET syntax

1. In the tag for the control, assign a method name to the name of the event. For example:

```
<input type="submit" value="Submit!"
   onServerClick="GreetMe" runat="server">
```

> **Note** **OnClick** is a client-side event, whereas **OnServerClick** is a server-side event for an HTML button.

2. Create an event-handling sub-procedure in your page code and access the properties of a control. The event handler takes two parameters as shown in the following sample code:

```
Sub GreetMe(s As Object, e As EventArgs)
   spnGreeting.InnerHTML = "Hello " & txtName.Value
End Sub
```

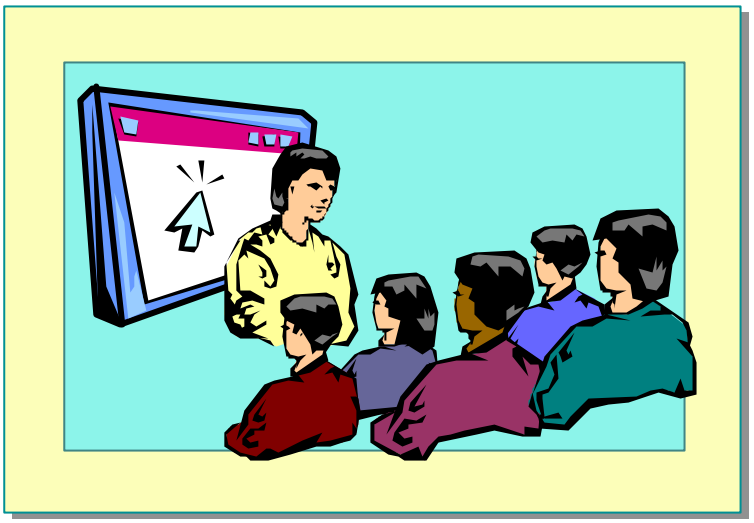
Event arguments

All events pass two arguments to the event procedure: the sender of the event, and an instance of the class that holds data for the event, if any. The second argument is usually of type **EventArgs** and often does not contain any additional information but, for some controls, it is of a type specific to that control.

For example, for an **ImageButton** Web control, the second argument is of the type **ImageClickEventArgs**, which includes information about the coordinates where the user clicked. The following event procedure outputs the coordinates of the click in a label control:

```
Sub img_OnClick(s As Object, e As ImageClickEventArgs)
  label1.Text = e.x & ", " & e.y
End Sub
```

Demonstration: Adding Code to Controls



In this demonstration, you will see how to add event procedures to an ASP.NET page.

The file <*install folder*>\Democode\Mod01\eventproc.aspx has the completed code for this demonstration.

↙ To run this demonstration

1. Open the file <*install folder*>\democode\Mod01\server_controls.aspx.
2. Add id attributes to the text box and the list box: **txtName** and **lstTitle**.
3. Add an **onServerClick** attribute to the button that will call the **GreetMe** procedure.

```
<input type="submit" onserverclick="GreetMe"
value="Save" runat="server">
```

4. Create a SCRIPT section for server-side code.

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

5. Create the **GreetMe** sub-procedure.

```
Sub GreetMe(s As Object, e As EventArgs)
   label1.Text = "Hello " & txtName.Value & _
       ". I see your occupation is " & lstTitle.Value
End Sub
```

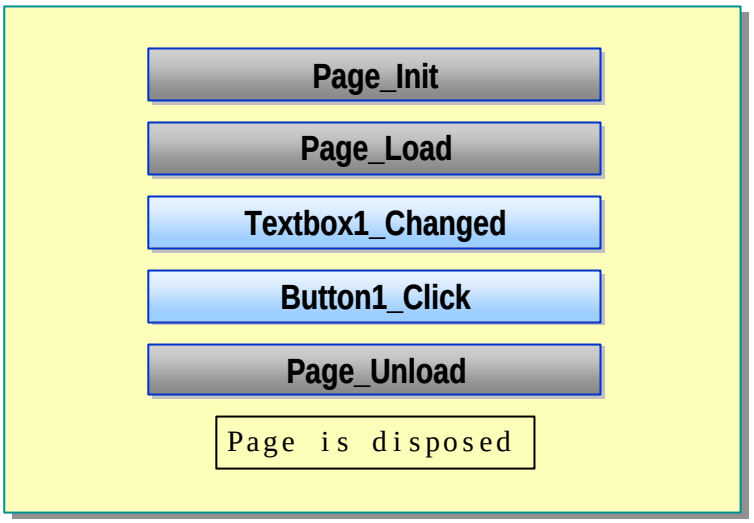
6. View the page in Internet Explorer. When you click the button, the values of the controls are displayed in the element.
7. View the source of the page. The code is server-side code so it is not included in the page.

Handling Page Events

- Page Event Life Cycle
- Handling Postback Forms
- Demonstration: Using Postback Forms

In this section, you will learn about the event life cycle of an ASP.NET page. You will learn about using the **Load** and **Unload** events of the **Page** object. Finally, you will learn how to use the **IsPostBack** property to handle postback forms.

Page Event Life Cycle



In ASP.NET, a structured set of events occurs every time a page is viewed. The event order is:

1. **Page_Init**

   After this event, a page is initialized. This event is responsible for all the initialization activities. If you want to initialize certain variables and so on, before a page loads, you declare it in the **Page_Init** event.

2. **Page_Load**

   The **Page_Load** event automatically fires every time a page is loaded. It fires at the beginning of a request after controls are initialized. Because server controls can be accessed on the server, you can load their data in the **Page_Load** event.

   ```
   Sub Page_Load(s As Object, e As EventArgs)
      txtName.Text = 'value from database
   End Sub
   ```

3. **Change** events

   After the **Page_Load** event, control-specific events are fired. An example of a control-specific event is a **Change** event or a **Click** event.

   By default, only **Click** events submit a form to the server. **Change** events are stored in a queue and handled in a random order on the server after the form is posted.

   For example, if a user enters text into a number of controls on a form and then clicks a submit button, the **Change** events for the text controls will not be processed until the form is sent to the server by the **Click** event.

4. **Click** event

   Whatever event caused the posting of the form fires after all **Change** events have completed.

5. **Page_Unload**

   The **Page_Unload** event is the last event fired before the page is discarded. This event is fired when the user goes to another page. You cannot read the values of controls in the **Page_Unload** event because by the time the event fires, the controls no longer exist. **Page_Unload** events are useful for cleanup activities such as logging, closing files, closing databases, and discarding objects.

```
Sub Page_Unload(s As Object,e As EventArgs)
   MyApp.LogPageComplete()
End Sub
```

Handling Postback Forms

- **ViewState Control Maintains the State of a Page During Postback**
- **Page_Load Fires on Every Request**
  - Use Page.IsPostBack to execute conditional logic

```
Sub Page_Load(s As Object, e As EventArgs)
  If Not Page.IsPostBack Then
    'executes only on initial page load
  End If
  'Rest of procedure executes on every request
End Sub
```

If you do not specify an action attribute on a form, ASP.NET pages post back to themselves. The **__VIEWSTATE hidden** control on a form maintains the state on the form so that the values previously entered do not disappear. This is very useful for form validation and building dynamic pages.

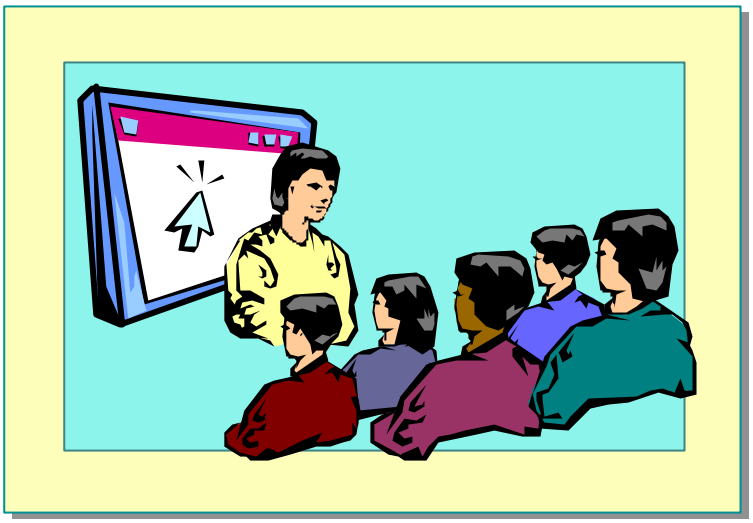
However, code in the **Page_Load** event runs each time the page is loaded. This code could re-execute data lookup to fill a list box, which is unnecessary, or override previous selections made by the user and maintained by the **__VIEWSTATE** control.

The **IsPostBack** property of the **Page** object is a Boolean value that is set to **True** whenever the page is posted back. This property can be used in the **Page_Load** event to specify which code in the **Page_Load** event run should run on every page load, and which should run only the first time the page is loaded.

For example, in the following code sample, the **IsPostBack** property is used to ensure that the text box is filled in only the first time.

```
Sub Page_Load(s As Object,e As EventArgs)
  If Not Page.IsPostBack Then
    'executes only on initial page load
    txtComment.Value = "initial value"
  End If
  'Rest of procedure executes on every request
End Sub
```

Demonstration: Using Postback Forms



In this demonstration, you will see how to use the **IsPostBack** property of an ASP.NET page.

The file <*install folder*>\Democode\Mod01\postback.aspx has the completed code for this demonstration.

↙ To run this demonstration

1. Open the file <*install folder*>\Democode\Mod01\server_controls.aspx.
2. Add a **Page_Load** event procedure that initializes the text box.

```
Sub Page_Load(s As Object, e As EventArgs)
   txtName.Value = "Enter your name"
End Sub
```

3. View the page in Internet Explorer. Enter text in the text box and click **Save**.

   The text box is loaded with default text.

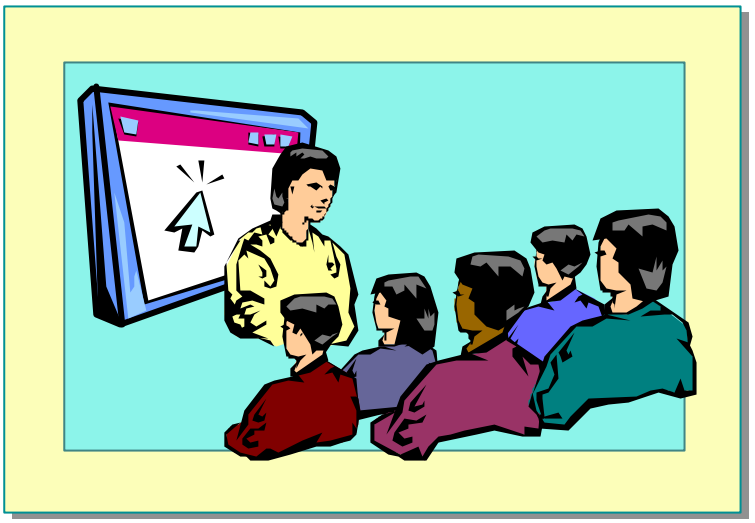
4. In the **Page_Load** event procedure, check for **IsPostBack** and initialize the text box only the first time the page is loaded.

```
If Not Page.IsPostBack Then
   txtName.Value = "Enter your name"
End If
```

5. View the page again. Enter text in the text box and click **Save**.

   Now the initial text is displayed only the first time the page is displayed.

Discussion: ASP vs. ASP.NET



Lab 1: Using ASP.NET to Output Text

Objectives

After completing this lab, you will be able to:

- Integrate HTML server controls in an ASP.NET page
- Write page event procedures and event procedure for server controls in an ASP.NET page

Prerequisite

Before working on this lab, you must know how to create an HTML page.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab01\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab01\Solution.

Scenario

A user must type an e-mail name and password in order to sign into the conference system. This is done on the login.aspx page. In this lab, you will first create an ASP.NET page with HTML server controls and then create event procedures for the button.

Estimated time to complete this lab: 60 minutes

Exercise 1
Creating an ASP.NET Page

In this exercise, you will create a simple ASP.NET page with a form, two HTML text boxes, an HTML button, and an HTML element. You will then create an event procedure for the button.

↙ To create the Web application

1. Using Microsoft Visual Studio .NET, create a new Web project named ASPNET.
   a. On the **File** menu, click **New**, and then click **Project**.
   b. In the **New Project** dialog box, click **ASP.NET Web Application**, set the **Name** to **ASPNET**, and then click **OK**.

      Visual Studio .NET creates a virtual root named ASPNET, and the following files:

      - aspnet.vbproj
      - aspnet.vbproj.webinfo
      - aspnet.vsdisco
      - AssemblyInfo.vb
      - Global.asax
      - Global.asax.resx
      - Global.asax.vb
      - Styles.css
      - Web.config
      - WebForm1.aspx
      - WebForm1.aspx.resx
      - WebForm1.aspx.vb
2. Add the starter lab files to the project.
   a. On the **Project** menu, click **Add Existing Item**.
   b. In the **Add Existing Item** dialog box, navigate to the <*install folder*>\Labs\Lab01\Starter folder.
   c. Select **All Files (*.*)** in the **Files of type** list box.
   d. Select all 19 files, and then click **Open**.

Note When copying the files, you will be asked if you want to replace the existing Web.config file. Click **Yes**. You will also be asked multiple times if you want to create a new class file. Click **No** for each new class file that you are asked to create.

---

e. Start Windows Explorer and navigate to the <*install folder*>\Labs\Lab01\Starter folder.

f. Select the bin, Components, and images folders and drag them to the root of the ASPNET Web site in the Visual Studio .NET **Solution Explorer** window.

Note When copying the folders, you will be asked if you want to replace the bin directory. Click **Yes**.

↙ To create a blank .aspx page

1. Open the WebForm1.aspx file.
2. Click the **HTML** tab (at the bottom of the page) to view the contents of the page.
3. Delete all attributes except the **Language** attribute from the @Page directive at the top of the page.

   The @Page directive should look like the following:

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

4. Save the page as login.aspx.

↙ To add a SCRIPT section

1. Add a SCRIPT section to the beginning of the login.aspx page (just before the <HTML> tag).
2. Set the **Language** attribute to Visual Basic.

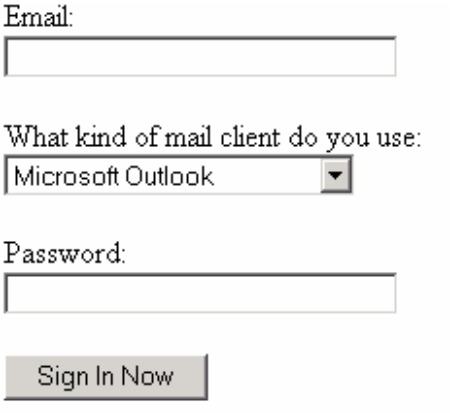
   Your script section should look the following:

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

   </script>
   ```

↙ To create the logon form

1. Add text and HTML server controls to the **default form** section of the file to make it look like the following illustration. Be sure to add a tag after the **Sign In Now** button.



Use the following table to create the controls:

Control	Attributes
E-mail	type = "text"
	id = "txtEmail"
Listbox (select)	id = "selMailClient"
Password	type = "password"
	id = "txtPassword"
Button	type = "submit"
	value = "Sign In Now"
Span	id = "spnInfo"

Your form should look like the following:

```
<form id="WebForm1" method="post" runat="server">
   Email:<br>
       <input type="text" id="txtEmail" runat="server">
   <br><br>
   What kind of mail client do you use:<br>
   <select id="selMailClient" runat="server">
   </select>
   <br><br>
   Password:<br><input type="password" id="txtPassword"
       runat="server"><br><br>
   <input type="submit" value="Sign In Now"
       runat="server">
   <span id="spnInfo" runat="server"> </span>
</form>
```

2. Fill the **listbox** server-control with the following options:
   - Microsoft Outlook®
   - Microsoft Outlook Express
   - Web-based
3. Make the first option the default selection.

   Your HTML for the list box should look the following:

```
<select id="selMailClient" runat="server">
   <option selected>Microsoft Outlook</option>
   <option>Microsoft Outlook Express</option>
   <option>Web-based</option>
</select>
```

↙ To add an event procedure for the button

1. Set the **onServerClick** attribute of the button to **cmdLogin_Click**.
2. In the <script> section, create the **cmdLogin_Click** event procedure.
3. Display the contents of the controls in the **spnInfo** SPAN element.

   To see all the comments written to the **spnInfo** SPAN element, concatenate the spnInfo.innerHTML content with the values from the other controls.

   Your event procedure should look like the following:

```
Sub cmdLogin_Click(s As Object, e As EventArgs)
   spnInfo.innerHTML = spnInfo.innerHTML & _
       "Your email: " & _
       txtEmail.Value & _
       "<br>Your mail client is: " & _
       selMailClient.Value & _
       "<br><br>"
End Sub
```

↙ To save and test your work

1. Save your changes.
2. Using Microsoft Internet Explorer, go to the login page of the ASPNET Web site by viewing http://localhost/ASPNET/login.aspx.
3. Fill the **Email** field with your e-mail address and click **Sign In Now**.

   You should see your e-mail address at the bottom of the page.

Exercise 2
Creating Page Event Procedures

In this exercise, you will add page event procedures to the page.

↙ To add a Page Load event procedure

1. In the script section, add a **Page_Load** event procedure that displays the value of **Page.IsPostBack**.

   Concatenate the value of the **IsPostBack** property with the current text in the **spnInfo** element.

   Your event procedure should look like the following:

```
Sub Page_Load(s As Object, e As EventArgs)
   spnInfo.innerHTML = spnInfo.innerHTML & _
       "One more Page_Load event!<br>" & _
       "Page.IsPostBack=" & _
       Page.IsPostBack.ToString() & _
       "<br><br>"
End Sub
```

2. Save your changes.
3. Using Internet Explorer, go to the logon page of the ASPNET Web site by viewing http://localhost/ASPNET/login.aspx.

   Notice the message on the page.
4. Click **Sign In Now**.

   Notice the postback message.

Review

- Introducing ASP.NET
- Creating Web Forms
- Adding ASP.NET Code to a Page
- Handling Page Events
- Discussion: ASP vs. ASP.NET

1. List some of the differences between ASP and ASP.NET.

2. What are the main features of ASP.NET?

3. What attribute is used to denote server controls?

4. How is the view state of controls maintained?

5. How do you create an event procedure for the click event of an HTML button control?

6. What is the order of events that occur when a page is displayed?