ASP.NET

M. MacDonald, Beginning ASP.NET 3.5 in C# 2008, 2nd Ed., 2007, Apress
M. MacDonald, M. Szpuszta, Pro ASP.NET 3.5 in C# 2008, 3rd Ed., 2009, Apress
S. Walther, ASP.NET 3.5 Unleashed, 2008, Sams
B. Evjen et al., Professional ASP.NET 3.5 SP1 Edition In C# and VB, 2009, Wrox
G. Shepherd, Microsoft ASP.NET 3.5 Step by Step, 2008, Microsoft Press
MSDN

Contents

General Information

ASP.NET Page and Controls Framework

- ASP.NET is a part of the .NET Framework
- A programming framework that runs on a Web server to dynamically produce and render ASP.NET Web pages
- ASP.NET Web pages can be requested from any browser or client device
  - ASP.NET renders appropriate markup for the browser making the request
  - It is also possible to target a specific browser
  - ASP.NET supports mobile controls for Web-enabled devices
- ASP.NET Web pages are completely object-oriented
  - There exist properties, methods, and events for all HTML elements
  - ASP.NET uses a unified model for responding to client events in code that runs on the server

ASP.NET Compilation

- All ASP.NET code is compiled
- The benefits from compiling:
  - Performance, security, stability, interoperability
- Features of the ASP.NET compilation architecture:
  - Multiple language support – different languages can be used in the same application
  - Automatic compilation – ASP.NET automatically compiles the application code and any dependent resources the first time a user requests a resource from the Web site
  - Flexible deployment
  - Extensible build system – custom classes to compile custom resources can be created

Security Infrastructure

- ASP.NET has all security features of the .NET Framework
- ASP.NET provides an advanced security infrastructure for authenticating and authorizing user access as well as performing other security-related tasks
    - Windows authentication supplied by IIS or ASP.NET forms authentication can be used
- ASP.NET always runs with a particular Windows identity
    - It is possible to secure the application using Windows capabilities, database permissions, and so on

State-Management Facilities

- ASP.NET provides intrinsic state management functionality
  - Information can be stored between page requests
- The following levels of state management are available:
  - Application-specific
  - Session-specific
  - Page-specific
  - User-specific
  - Developer-defined
- ASP.NET offers distributed state facilities
  - It allows to manage state information across multiple instances of the same application on several computers

ASP.NET Configuration

- ASP.NET configuration system allows to define configuration settings for a Web server, Web site, or for individual applications
  - Configuration settings can be revised at any time with minimal impact on operational Web applications and servers
- ASP.NET configuration settings are stored in XML-based files
  - **`machine.config`**
  - **`Web.config`**

Health Monitoring and Performance Features

- ASP.NET provides an easy way to monitor the health of deployed ASP.NET applications
  - It provides detailed run-time information about ASP.NET resources
- ASP.NET supports two groups of performance counters accessible to applications:
  - The ASP.NET system performance counter group
  - The ASP.NET application performance counter group

Debugging Support

- ASP.NET provides cross-language and cross-computer debugging support
  - Both managed and unmanaged objects can be debugged
  - All CLR languages and script languages can be debugged
- Trace mode supported by ASP.NET enables to insert instrumentation messages into ASP.NET Web pages
  - Trace messages can be accessed programmatically

ASP.NET 3.5 New Features

- ASP.NET 3.5 specific features:
  - The ListView control
  - The DataPager control
  - The LinqDataSource
  - Integrated AJAX support
  - Improved designer support for AJAX control extenders
- ASP.NET 3.5 - related Visual Studio 2008 enhancements
  - Nested Master Page support
  - Multi-Framework Targeting
  - Enhanced Web design experience
  - JavaScript IntelliSense
  - JavaScript debugging
  - CSS Properties window
  - Manage Styles tool window

ASP.NET Development Requirements

- .NET Framework
- Code authoring environments (e.g Visual Studio or Visual Web Developer)
- Web servers
  - Visual Studio uses the ASP.NET Development Server which runs pages locally without requiring to install IIS
  - ASP.NET applications are typically hosted using IIS (Internet Information Services) as the Web server
- Optionally:
  - Databases
    - Some ASP.NET features such as membership and profile properties, require a database
  - SMTP servers
    - Some ASP.NET controls, such as `PasswordRecovery`, require the ability to send e-mail messages

Application Folders

- **`App_Browsers`** - browser definitions
- **`App_Code`** - source code for utility classes and business objects that should be compiled
- **`App_Data`** - application data files (.mdf, .xml, etc.)
- **`App_GlobalResources`** - resources (**`.resx`** and **`.resources`** files) of global scope
- **`App_LocalResources`** – resources associated with a specific page, user control, or master page in the application
- **`App_Themes`** - collection of files defining the appearance of ASP.NET Web pages and controls
- **`App_WebReferences`** - reference contract files (**`.wsdl`** files), schemas (**`.xsd`** files), and discovery document files (**`.disco`** and **`.discomap`** files) defining a Web reference
- **`Bin`** - compiled assemblies (**`.dll`** files) for controls, components, or other code referenced in the application

Web Site File Types

- `.asax` - typically a `Global.asax` file with application's code
- `.ascx` - web user control files that defines a custom, reusable user controls
- `.ashx` - generic handler files that contains code to handle all incoming requests
- `.asmx` - XML Web Service files with classes and methods
- `.aspx` - ASP.NET Web forms file that can contain Web controls and other business logic
- `.axd` - trace-viewer rules, typically `Trace.axd`
- `.browser` - browser definition files used to identify the enabled features of the client browser
- `.compile` - precompiled stub files that points to the appropriate assembly (e.g for `.aspx`, `.ascx`, `.master`)
- `.cs`, `.jsl`, `.vb` - class source-code files

Web Site File Types cont.

- `.css` - style sheet files
- `.csproj, .vbproj, .vjsproj, .sln` - Visual Studio project and solution files
- `.disco, .vsdisco` - XML Web Services Discovery files
- `.dll` - compiled class library files (alternatively source code for classes can be put in the `App_Code` subdirectory)
- `.licx, .webinfo` - licence files
- `.master` - master page files
- `.mdb, .ldb` - Access database files
- `.mdf` - SQL Server database files
- `.resources, .resx` – resource files
- `.sitemap` - site-map file with the structure of the Web site
- `.skin` - skin files used to determine display formatting

Specifying Paths for Resources

- Client elements

```
<img src="http://www.abc.com/MyApplication/Images/Image.jpg" />
<img src="/Images/SampleImage.jpg" />  - a site root relative path
<img src="Images/SampleImage.jpg" />   - a relative path
<img src="../Images/SampleImage.jpg" />
```

- Server controls

```
<asp:image runat="server" id="Image1"
           ImageUrl="~/Images/SampleImage.jpg" />
```

- The **`HttpRequest.MapPath()`** method returns the complete physical path for a virtual path

```
String rootPath = Server.MapPath("~");
```

```
a request from a browser:
http://www.contoso.com/MyApplication/MyPages/Default.aspx
the physical path for the root of the Web site:
C:\inetpub\wwwroot\MyApplication
```

- Results of **`HttpRequest`** properties:
  - **`ApplicationPath`** (root path of the current application)
    `/`
  - **`CurrentExecutionFilePath`** (correct also after redirection in server code)
    `/MyApplication/MyPages/Default.aspx`
  - **`FilePath`** (virtual path of the current request)
    `/MyApplication/MyPages/Default.aspx`
  - **`Path`** (includes the PathInfo trailer)
    `/MyApplication/MyPages/Default.aspx`
  - **`PhysicalApplicationPath`** (physical path of the application's root)
    `C:\inetpub\wwwroot\`
  - **`PhysicalPath`** (physical path of the requested URL)
    `C:\inetpub\wwwroot\MyApplication\MyPages\default.aspx`

ASP.NET Application Life Cycle

1. User requests an application resource from the Web server
2. ASP.NET receives the first request for the application
   - ASP.NET compiles the top-level items in the application if required, including application code in the **`App_Code`** folder
3. ASP.NET core objects are created for each request
   - **`HttpContext`**, **`HttpRequest`**, and **`HttpResponse`**
4. An **`HttpApplication`** object is assigned to the request
   - The application is started by creating an instance of the **`HttpApplication`** class, this instance is used for all requests
5. The request is processed by the **`HttpApplication`** pipeline

Key Members of HttpContext Class

- `ApplicationInstance` (type: `HttpApplication`)
- `PreviousHandler` (`HttpHandler`) – an instance of the handler that rendered the previous page (used in cross-page postbacks); it is exposed at the page level as the `PreviousPage` property
- `Request` (`HttpRequest`) – the request object
- `Response` (`HttpResponse`) – the response object
- `Server` (`HttpServerUtility`) – a few useful methods
- `Application` (type: `HttpApplicationState`) – stores application specific data, shared for all users
- `Session` (`HttpSessionState`) – session of the user
- `Profile` (`HttpProfileBase`) – personalisation data
- `Cache` (`Cache`) – the ASP.NET cache object

Pages

Code-Inline Model

```
<%@ Page Language="C#"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Button pushed";
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
      Text="Button" />
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
  </div>
  </form>
</body>
</html>
```

Code-Behind Model

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="WebForm1.aspx.cs" Inherits="TespApp.WebForm1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
  </div>
  </form>
</body>
</html>
```

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace TespApp {
    public partial class WebForm1 : System.Web.UI.Page {
        protected void Button1_Click(object sender, EventArgs e) {
            Label1.Text = "Button clicked";
        }
    }
}
```

ASP.NET Directives

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="WebForm1.aspx.cs" Inherits="CodeInline.WebForm1" %>
```

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeBehind="Site1.master.cs" Inherits="CodeInline.Site1" %>
```

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="WebUserControl1.ascx.cs"
Inherits="TestApp.WebUserControl1" %>
```

- Directives are used to pass optional settings to the ASP.NET pages and compilers
- Directives are typically located at the top of the appropriate file though that is not a strict requirement

List of Directives

- **`Application`** - application-specific attributes
- **`Assembly`** - links an assembly to the application or page
- **`Control`** - contained in user control files
- **`Implements`** - a COM interface implemented by the page
- **`Import`** - imports a namespace into a page, user control, or application, making all of the classes and namespaces of the imported namespace available
- **`Master`** - identifies a page file as being a master page
- **`MasterType`** - assigns a class name to the **`Master`** property of the page
- **`OutputCache`** - controls output caching for a page or user control
- **`Page`** - attributes for the page parser and compiler specific
- **`Reference`** - indicates that another page or user control should be compiled and linked to the current page
- **`Register`** - associates aliases with namespaces (used in custom server controls and user controls)

Common Page Directive Attributes

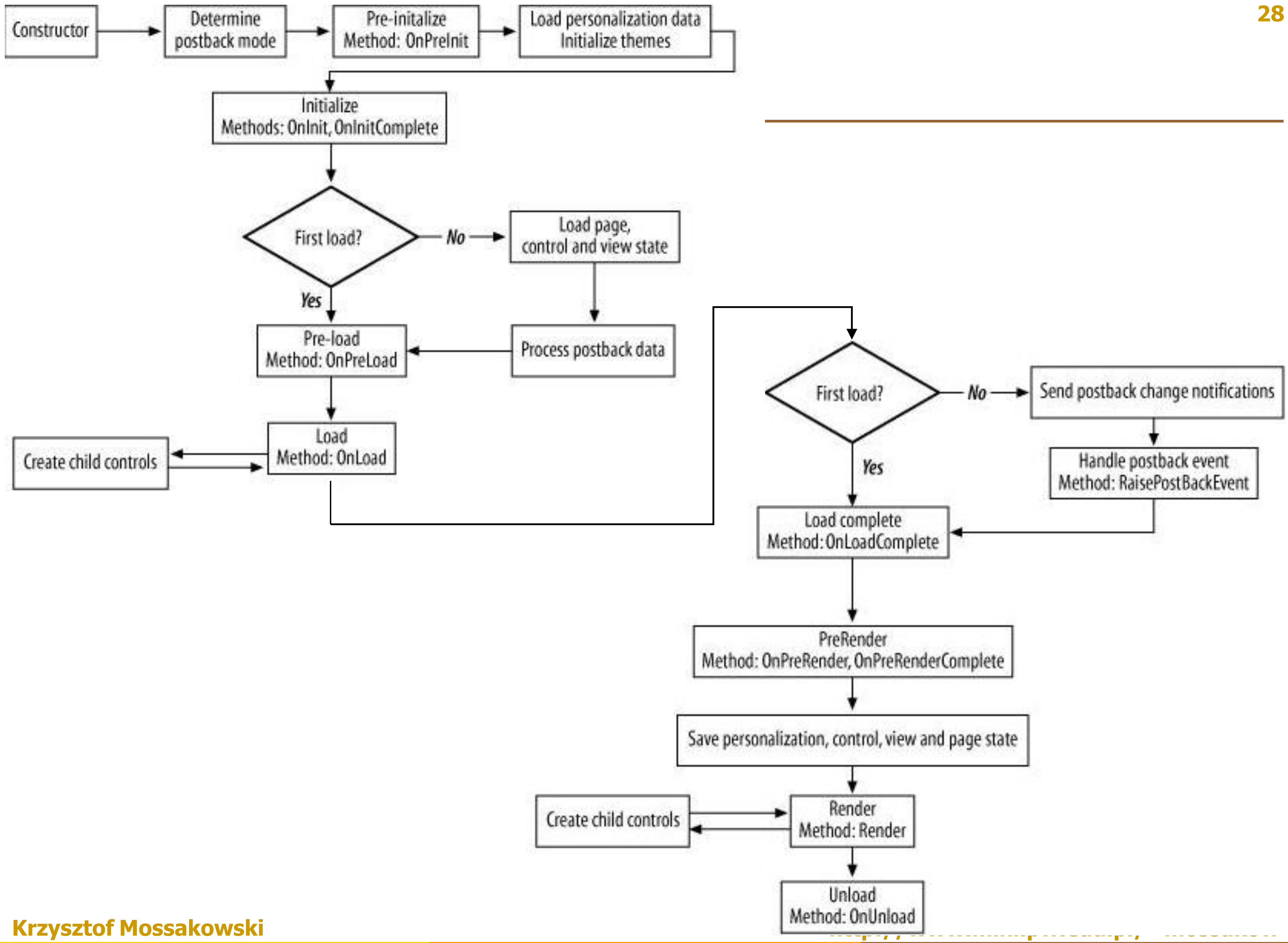
- **`AutoEventWireup`** - enables or disables page events being automatically bound to methods that follow the naming convention Page_*event* (the default is **`true`**)
- **`Buffer`** - enables or disables the HTTP response buffering
- **`ClientTarget`** - targets the user agent that server controls should render content for
- **`CodeFile`** - used by Visual Studio to indicate the name of the code-behind file
- **`Debug`** - enables or disables compiling with debug symbols (the default is **`false`**)
- **`Description`** - text description of the page; ignored by the parser
- **`EnableSessionState`** - enables, disables, or makes **`SessionState`** read-only (the default is **`true`**)
- **`EnableViewState`** - enables or disables maintenance of view state across page requests (the default is **`true`**)

Common Page Directive Attributes cont.

- **`ErrorPage`** - targets URL for redirection if an unhandled page exception occurs
- **`Inherits`** - name of the code-behind or other class
- **`Language`** – the programming language used for in-line code
- **`SmartNavigation`** - indicates support for smart navigation by the browser, which enables scroll position on a page to survive postbacks (the default is **`false`**)
- **`Src`** - relative or fully qualified filename containing the code behind class
- **`Trace`** - enables or disables tracing (the default is **`false`**)
- **`TraceMode`** - indicates how trace messages are to be displayed
- **`Transaction`** - indicates whether transactions are supported on the page
- **`ValidateRequest`** - if **`true`** (the default) all input data is validated against a hard-coded list of potentially dangerous values

ASP.NET Page Life Cycle

1. A page request
2. Start - page properties are set (e.g. `Request`, `Response`, `IsPostBack`)
3. A page initialization - controls on the page are available and each control's `UniqueID` property is set
4. Load - if the current request is a postback, control properties are loaded with information recovered from view state and control state
5. Validation - the `Validate()` methods of every event handler is called , which set the `IsValid` property of individual validation controls and of the page
6. Postback event handling - if the request is a postback, any event handlers are called
7. Rendering – saving view state to the page, rendering each control to the `OutputStream` of the page's `Response` property
8. Unload – unloading the page's properties, final cleanup


Constructor
Determine postback mode
Pre-initalize Method: OnPreInit
Load personalization data Initialize themes
Initialize Methods: OnInit, OnInitComplete
First load?
No
Load page, control and view state
Yes
Pre-load Method: OnPreLoad
Process postback data
Create child controls
Load Method: OnLoad
First load?
No
Send postback change notifications
Yes
Handle postback event Method: RaisePostBackEvent
Load complete Method: OnLoadComplete
PreRender Method: OnPreRender, OnPreRenderComplete
Save personalization, control, view and page state
Create child controls
Render Method: Render
Unload Method: OnUnload

Typical Use of Page Events

- **`Page_PreInit`**
  - Use the **`IsPostBack`** property to determine whether this is the first time the page is being processed
  - Create or re-create dynamic controls
  - Set a master page dynamically
  - Set the **`Theme`** property dynamically
  - Read or set profile property values
  - Do not set controls properties as their values might be overwritten in the next stage
- **`Page_Init`**
  - Read or initialize control properties
- **`Page_InitComplete`**
- **`Page_PreLoad`**
- **`Page_Load`**
  - Read and update control properties

Typical Use of Page Events cont.

- **`Page_LoadComplete`**
- Control events
  - If the page contains validation controls, check the **`IsValid`** property of the page and of individual validation controls before performing any processing
  - Handle the specific event, such as a **`Button`** control's **`Click`** event
- **`Page_PreRender`**
  - Make final changes to the contents of the page
- **`Page_PreRenderComplete`**
- **`Page_Unload`**
  - Perform final cleanup work (e.g. close open files and database connections, finish logging or other request-specific tasks)
  - Do not make any changes to the response stream (it will throw an exception)

Postbacks

- ASP.NET pages typically post back to themselves in order to process events (such as a button-click event)
  - For this reason, you must differentiate between posts for the first time a page is loaded by the end user and postbacks
- A postback – a posting back to the same page
  - A postback contains all the form information collected from the initial page
- The `IsPostBack` property can be used to check whether a request is the first instance for a particular page or a postback from the same page

Cross-Page Posting

- Posting to another page and dealing with the source page's control values on that page
  - Controls from the first page can be found in source code of the second page using the **`PreviousPage.FindControl()`** method
  - All public properties of the first page are available for the second page
  - The second page can check the **`IsCrossPagePostBack`** property

```
<asp:Textbox ID="TextBox1" runat="server"></asp:Textbox>
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
       Text="Submit page to itself" />
<asp:Button ID="Button2" runat="server" PostBackUrl="WebForm2.aspx"
       Text="Submit page to Page2.aspx" />
```

```
TextBox tb = (TextBox)PreviousPage.FindControl("TextBox1");
Label1.Text = tb.Text;
```

Moving to Another Page

- The **`Hyperlink`** control navigates directly to the location contained in the **`NavigateUrl`** property of the control without a postback to the server
- The **`HttpServerUtility.Transfer()`** method takes an URL of an .aspx or .html page as a string argument and posts back to the server
  - It raises the **`ThreadAbortException`**
- The **`HttpResponse.Redirect()`** method is the programmatic equivalent of a **`Hyperlink`**
  - It is a completely new server request, it forces complete authentication and authorization
- Cross-page posting

Page Compilation

- The first request to the page:
  1. The parser creates the class file in the language of the page
  2. The class is compiled into a DLL and written to the disk of the Web server
  3. The DLL is instantiated and processed
  4. Output is generated
- The next request:
  1. The DLL is instantiated and processed
  2. Output is generated
- Changes made on the page will cause recompilation when the page is requested again
- In ASP.NET 2.0 a special tool for full precompilation was introduced

```
http://[host]:[port]/[Application Name]/precompile.axd
```

The Global.asax File

- **`Global.asax`** (the Global Application Class file) is used by the application to hold application-level events, objects, and variables
- Events available in this file:
  - **`Application_Start`**
  - **`Session_Start`**
  - **`Application_BeginRequest`** – triggered before each request
  - **`Application_AuthenticateRequest`** – triggered for each request, enables to set up custom authentication
  - **`Application_Error`**
  - **`Session_End`**
  - **`Application_End`**

Commonly Used Page Properties

- **`Application`** - reference to the **`Application`** object
- **`Cache`** - the **`Cache`** object of the application
- **`ClientQueryString`** - the query string portion of the URL
- **`Controls`** - a reference to the collection of controls
- **`ErrorPage`** - the name of the page for error redirecting
- **`IsCrossPagePostBack`** - if **`true`**, this page called another page using the cross-page postback
- **`IsPostBack`** - if **`true`**, this page is being loaded in response to a client post
- **`IsValid`** - if **`true`**, validation succeeded for all the controls
- **`MaintainScrollPositionOnPostback`** - if **`true`**, the browser positioning of the page will be preserved across postbacks (the default is **`false`**)
- **`Master`** - retrieves a reference to the master page for the page
- **`MasterPageFile`** - the filename of the master page
- **`PreviousPage`** - retrieves a reference to the previous page

Controls

Types of Web Controls

- HTML controls
  - The original controls available to any HTML page
- HTML server controls
  - Based on the original HTML controls but enhanced to enable server-side processing
- ASP.NET server controls (aka Web server controls)
  - Rich and flexible server-side controls, integrated into the ASP.NET programming mode
  - These controls are rendered to the client as HTML code and provide the same functionality as HTML server controls and more
  - Can be declared in a content file or programmatically instantiated and manipulated in .NET assemblies
- User controls and custom controls
  - Controls created by developers

Controls
HTML Server Controls

HTML Server Controls

- Normal HTML controls such as `<h1>`, `<a>`, and `<input>` are not processed by the server, but are sent directly to the browser for display
  - They can be exposed to the server and made available for server-side processing by turning them into HTML server controls
  - A normal HTML control can be converted into an HTML server control by adding the `runat` parameter

```
<input type="text" size="40">
```

```
<input type="text" id="BookTitle" size="40" runat="server">
```

Usage of the HTML Server Controls

- Converting existing HTML pages to run under ASP.NET
  - It is enough to change the extension of the file to .aspx
  - The `runat` attribute must be added to take advantage of server-side processing (including automatic maintenance of the state)
- Using HTML tables for the table layout
  - For static tables commonly used to lay out the page, server-side processing is unnecessary

HTML Server Controls Classes

- `<head>` - `HtmlHead`
- `<input>` - `HtmlInputButton`, `HtmlInputCheckbox`, `HtmlInputFile`, `HtmlInputHidden`, `HtmlInputImage`, `HtmlInputPassword`, `HtmlInputRadioButton`, `HtmlInputReset`, `HtmlInputSubmit`, `HtmlInputText`
- `<img>` - `HtmlImage`
- `<link>` - `HtmlLink`
- `<textarea>` - `HtmlTextArea`
- `<a>` - `HtmlAnchor`
- `<button>` - `HtmlButton`
- `<form>` - `HtmlForm`
- `<table>` - `HtmlTable`
- `<td>`, `<th>` - `HtmlTableCell`
- `<tr>` - `HtmlTableRow`
- `<title>` - `HtmlTitle`
- `<select>` - `HtmlSelect`
- `HtmlGenericControl`

Server-Side and Client-Side Processing

- Server-side processing (the heart of ASP.NET) needs a postback to the server
  - Even for intranet applications connected to the server with a high-speed local network connection, this introduces a noticeable delay often unacceptable
- Client-side processing can greatly enhance the user experience
  - It provides nearly instantaneous response to the user's actions
- Some ASP.NET server controls use client-side scripting to provide responses to user actions without posting back to the server (e.g. validation controls)
  - ASP.NET provides client-side scripts for these controls

```
<script language="javascript">
function ButtonTest() {
    alert("Button clicked - client side processing");
}
function DoChange() {
    document.getElementById("btnSave").disabled = false;
}
</script>

<input id="btnHTML" runat="server" type="button" value="HTML Button"
    onclick="javascript:ButtonTest();"
    onserverclick="btnHTML_ServerClick" />
<asp:Button ID="btnServer" runat="server" Text="ASP.NET Button"
    OnClientClick="javascript:ButtonTest();" />
<input id="txtHTML" type="text" runat="server"
    onchange="javascript:DoChange();" /><br />
<asp:TextBox ID="TextBox1" runat="server"
    onchange="javascript:DoChange();" />
<asp:Button ID="btnSave" runat="server" Text="Save" Enabled="false" />
```

```
protected void btnHTML_ServerClick(object sender, EventArgs e){
    txtHTML.Value = "An HTML server control";
}
```

Controls
ASP.NET Server Controls

ASP.NET Server Controls Advantages

- The ability to have the page automatically maintain the state of the control
- ASP.NET detects the level of the target browser, the appropriate HTML is generated for each browser
- The use of a compiled language instead of an interpreted script results in better performance
- The ability to bind to a data source
- Events can be raised by controls on the browser and easily handled by code on the server

Categories of ASP.NET Server Controls

- Basic controls
- Advanced controls
- Validation controls
- AJAX controls
- Data source controls
- Data view controls (data-bound controls)
- Web parts controls
- Site navigation controls
- Login and security controls

Basic Controls

- `Label, Literal`
- `TextBox`
- `Button, LinkButton, ImageButton`
- `HyperLink`
- `DropDownList, ListBox`
- `CheckBox, RadioButton`
- `CheckBoxList, RadioButtonList, BulletedList,`
- `Image, ImageMap`

Advanced Controls

- `Table`
- `Calendar`
- `AdRotator`
- `FileUpload`
- `Panel`
- `MultiView, View`
- `Wizard`
- `PlaceHolder`
- `HiddenField`
- `Substitution`
- `Xml`

Controls
Validation Controls

Validation Controls

- Validation controls provide an easy-to-use mechanism for all common types of standard validation plus ways to provide custom-written validation
- Validation controls allow to customize how error information is displayed to the user
- They can work with any controls put on the ASP.NET Web page, including both HTML and ASP.NET server controls
- Controls:
  - **`RequiredFieldValidator`**
  - **`CompareValidator`**
  - **`RangeValidator`**
  - **`RegularExpressionValidator`**
  - **`CustomValidator`**
  - **`ValidationSummary`**

Client-Side and Server-Side Validation

- Client-side validation
  - Quick and responsive for the end user
  - If something is wrong with the form, using client-side validation ensures that the end user knows this as soon as possible
  - It can be easily hacked or disabled, because client-side validation code is written in JavaScript and is fully visible for the end user
- Server-side validation
  - Requires a postback to the server
  - Much slower, but definitely more secure
- The best approach is always to perform client-side validation first and then, after the form passes and is posted to the server, to perform the validation checks again using server-side validation

Validation Controls

- Validation controls provide an easy-to-use mechanism for all common types of standard validation plus ways to provide custom-written validation
- Validation controls allow to customize how error information is displayed to the user
- They can work with any controls put on the ASP.NET Web page, including both HTML and ASP.NET server controls

Client-Side and Server-Side Validation

- Client-side validation
  - Quick and responsive for the end user
  - If something is wrong with the form, using client-side validation ensures that the end user knows this as soon as possible
  - It can be easily hacked or disabled, because client-side validation code is written in JavaScript and is fully visible for the end user
- Server-side validation
  - Requires a postback to the server
  - Much slower, but definitely more secure
- The best approach is always to perform client-side validation first and then, after the form passes and is posted to the server, to perform the validation checks again using server-side validation

ASP.NET Validation Server Controls

- ASP.NET performs browser detection when generating the ASP.NET page
  - If the browser can support the JavaScript that ASP.NET can send its way, the validation occurs on the client-side
  - It can be turned off using the **`EnableClientScript`** property
- Even if the client-side validation is initiated on a page, ASP.NET still performs the server-side validation when it receives the submitted page
- Custom validation controls can be created
- Validation occurs in response to an event (in most cases, it is a button click event)
  - The **`CausesValidation`** property of the **`Button`**, **`LinkButton`**, and **`ImageButton`** server controls is set to **`true`** by default

Validation Controls

- `RequiredFieldValidator`
  - Ensures that the user does not skip a form entry field
- `CompareValidator`
  - Allows for comparisons between the user’s input and another item using a comparison operator (equals, greater than, less than, etc.)
- `RangeValidator`
  - Checks the user’s input based upon a range of numbers or characters
- `RegularExpressionValidator`
  - Checks that the user’s entry matches a pattern defined by a regular expression (useful for e-mail addresses or phone numbers)
- `CustomValidator`
  - Checks the user’s entry using custom-coded validation logic
- `ValidationSummary`
  - Displays all the error messages from the validation controls in one specific spot on the page

Validation Groups

- The **`ValidationGroup`** property allows to separate the validation controls into groups
  - It allows to activate only the required validation controls when the end user clicks a button on the page

```
<asp:Button ID="Button1" Runat="server" Text="Login"
            ValidationGroup="Login" />
<br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
            Runat="server"
            ErrorMessage="* You must submit a username!"
            ControlToValidate="TextBox1"
            ValidationGroup="Login">
</asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
            Runat="server"
            ErrorMessage="* You must submit a password!"
            ControlToValidate="TextBox2"
            ValidationGroup="Login">
</asp:RequiredFieldValidator>
```

ASP.NET AJAX
Asynchronous JavaScript and XML

AJAX Overview

- AJAX includes some new client-script libraries to facilitate the asynchronous calls back to the server
  - Instead of the browser simply rendering streams of HTML and executing small custom-written script blocks
- AJAX also includes some basic server-side components to support these new asynchronous calls coming from the client

ASP.NET AJAX

Client Side **The AJAX Library**	**Server Side** **ASP.NET Extensions for AJAX**
Components Nonvisual components Behaviours, Controls	**Scripting** Localization, Globalization, Debugging, Tracing
Browser Compatibility Support for browsers: Microsoft Internet Explorer, Mozilla Firefox, Opera, Apple Safari	**Web Services** Proxy Generation, Page Methods, XML and JSON Serialization
Networking Asynchronous requests, XML and JSON Serialization, Web and Application Services	**Application Services** Authentication and profile support
Core Services JavaScript, Base Client Extensions, Type System, Events, Serialization	**Server Controls** ScriptManager, Update Panel, Update Progress, Timer

Reasons to Use AJAX

- AJAX improves the overall efficiency of a web site by performing parts of a Web page's processing in the browser when appropriate
  - ASP.NET’s AJAX support includes a number of scripts so that you can get a lot of browser-based efficiency by simply using a few server-side controls
- AJAX introduces UI elements usually found in desktop applications to a Web site
- AJAX introduces partial-page updates
- AJAX is supported by most popular browsers
- AJAX introduces a huge number of new capabilities using the extender control
- AJAX improves on ASP.NET’s forms authentication and profiles and personalization services

ASP.NET Server Side Controls

- **`ScriptManager`**
  - It manages script resources for the page
  - Its primiary action is to register the AJAX Library script with the page so the client script may use type system extensions
  - It makes possible partial-page rendering and supports localization as well as custom user scripts.
  - Any ASP.NET site wishing to use AJAX must include an instance of the ScriptManager control on any page using AJAX functionality
- **`ScriptManagerProxy`**
  - Nested components such as content pages and User controls require this control to manage script and service references to pages that already have a **`ScriptManager`** control
  - This is most notable in the case of master pages: there should be the **`ScriptManager`** control on the master page and the **`ScriptManagerProxy`** controls on the content pages

ASP.NET Server Side Controls cont.

- `UpdatePanel`
  - It supports partial page updates by tying together specific server-side controls and events that cause them to render
  - It causes only selected parts of the page to be refreshed instead of refreshing the whole page
- `UpdateProgress`
  - It coordinates status information about partial-page updates as they occur within `UpdatePanel` controls
  - It supports intermediate feedback for long-running operations
- `Timer`
  - It will issue postbacks at defined intervals
  - Although it will perform a normal postback (posting the whole page), it is especially useful when coordinated with the `UpdatePanel` control to perform periodic partial-page updates

ASP.NET AJAX Control Toolkit

- The ASP.NET AJAX Control Toolkit is a shared source project built on top of the Microsoft ASP.NET AJAX framework
  - http://www.asp.net/ajax/ajaxcontroltoolkit/
- It is a collection of components encapsulating AJAX's capabilities
  - http://www.asp.net/ajax/ajaxcontroltoolkit/samples/
  - Accordion, AlwaysVisibleControl, Animation, AsyncFileUpload, AutoComplete, Calendar, CascadingDropDown, CollapsiblePanel, ColorPicker, ComboBox, ConfirmButton, DragPanel, DropDown, DropShadow, DynamicPopulate, FilteredTextBox, HoverMenu, HTMLEditor, ListSearch, MaskedEdit, ModalPopup, MultiHandleSlider, MutuallyExclusiveCheckBox, NoBot, NumericUpDown, PagingBulletedList, PasswordStrength , PopupControl, Rating, ReorderList, ResizableControl, RoundedCorners, Seadragon, Slider, SlideShow, Tabs, TextBoxWatermark, ToggleButton, UpdatePanelAnimation, ValidatorCallout
- It also provides a powerful software development kit for creating custom controls and extenders

Data Binding

ASP.NET Data Access

- The `System.Data` (ADO.NET) and `System.Xml` namespaces can be used to write code to access data
- ASP.NET allows to perform data binding declaratively, no code is required for the most common data scenarios:
  - Selecting and displaying data
  - Sorting, paging, and caching data
  - Updating, inserting, and deleting data
  - Filtering data using run-time parameters
  - Creating master-detail scenarios using parameters
- Two types of server controls participate in the declarative data binding model:
  - Data source controls
  - Data-bound controls

Data Source Controls

- Data source controls are ASP.NET controls that manage the tasks of connecting to a data source, reading and writing data
  - They do not render any user interface
  - They act as an intermediary between a particular data store and other controls on the ASP.NET Web page
  - They enable rich capabilities for retrieving and modifying data, including querying, sorting, paging, filtering, updating, deleting, and inserting

Data Source Controls

- `SqlDataSource` - for any SQL database; it returns data as `DataReader` or `DataSet` objects
- `AccessDataSource` - specialized version of the `SqlDataSource` control - only for Microsoft Access
- `LinqDataSource` - for LINQ to SQL
- `EntityDataSource` (3.5 SP1) - for Entity Framework
- `XmlDataSource` - for data stored in XML files
- `SiteMapDataSource` - special data source for a definition of web site's structure; used by navigational controls
- `ObjectDataSource` - for special, custom logic of loading data (e.g. from services)

Data-Bound Controls

- `GridView` – displays tabular data
- `DetailsView` – displays a single record from a data source (each data row represents a field in the record)
- `FormView` – works like the DetailsView, but it displays the data in custom templates
- `Repeater` – produces a list of individual items
- `DataList` – displays rows of database information in customizable format
- `ListView` – displays data in a format defined by using templates and styles
  - it is similar to the DataList and Repeater controls, but it is possible to allow the user to modify displayed data without code
- `DataPager` – is used to page data and to display navigation controls for data-bound controls (e.g. the ListView control)

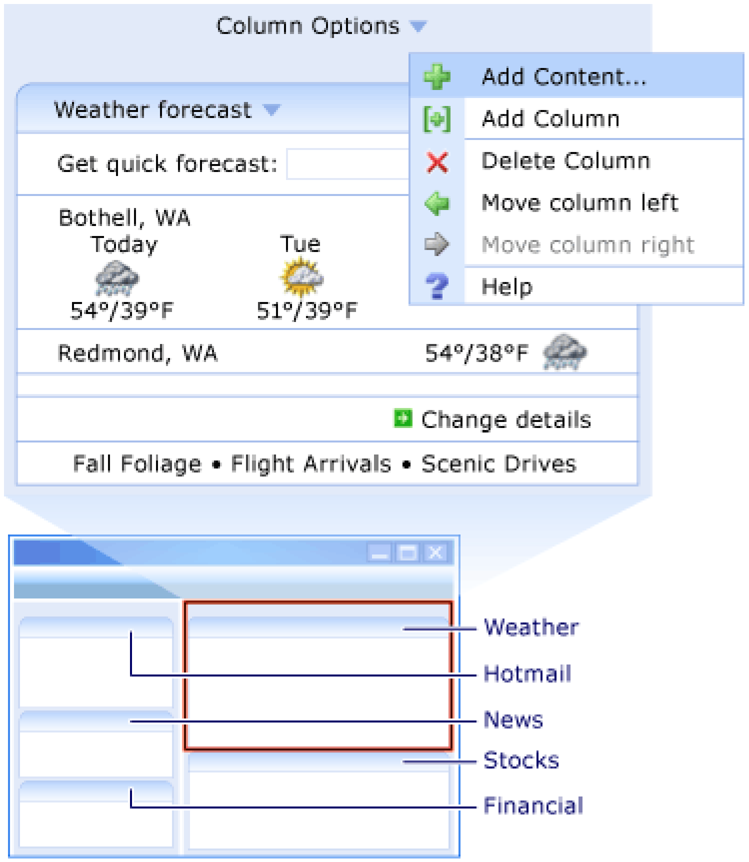
Web Parts

Web Parts

- ASP.NET Web Parts is an integrated set of controls for creating Web sites
- Web Parts enable end users to modify the content, appearance, and behaviour of Web pages directly from a browser
  - The modifications can be applied to all users on the site or to individual users
  - When a user modifies pages and controls, the settings can be saved to retain the user's personal preferences across future browser sessions (using the personalisation)

Possibilities for Users

- Web Parts allow end users to:
  - Personalise the page content
  - Personalise the page layout
  - Export and import controls
  - Create connections between controls
  - Manage and personalise site-level settings



WebParts Controls

- `WebPartManager, ProxyWebPartManager`
- `WebPartZone`
- `CatalogZone`
  - `DeclarativeCatalogPart`
  - `PageCatalogPart`
  - `ImportCatalogPart`
- `EditorZone`
  - `AppearanceEditorPart`
  - `BehaviorEditorPart`
  - `LayoutEditorPart`
  - `PropertyGridEditorPart`
- `ConnectionsZone`

Site Navigation

Site Navigation

- Site navigation features can be used to provide a consistent way for users to navigate the Web site
  - It enables to store links to all pages in a central location and render these links in lists or navigation menus on each page by including a specific Web server control

Features of Site Navigation

- Site maps can be used to describe the logical structure of the site
- ASP.NET controls can be used to display navigation menus on the Web pages
  - The navigation menu is based on the site map
- The site navigation can be controlled programmatically
- Access rules can be configured that display or hide a link in the navigation menu
- Custom site-map providers can be created
  - For example, a database where link information is stored

Site Maps

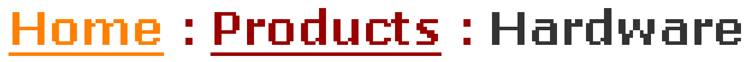
- By default, the site navigation system uses an XML file that contains the site hierarchy
  - The site navigation system can be configured to use alternative data sources
  - The simplest way to create a site map is to create an XML file named **`Web.sitemap`** that organizes the pages in the site hierarchically
- More than one site-map file or provider can be used to describe the navigation structure of the entire Web site

Example of the Web.sitemap file

```
<siteMap>
    <siteMapNode title="Home"
               description="Home"
               url="~/default.aspx">
        <siteMapNode title="Products"
                     description="Our products"
                     url="~/Products.aspx">
            <siteMapNode title="Hardware"
                         description="Hardware choices"
                         url="~/Hardware.aspx" />
            <siteMapNode title="Software"
                         description="Software choices"
                         url="~/Software.aspx" />
        </siteMapNode>
        <siteMapNode title="Services"
                 description="Services we offer"
                 url="~/Services.aspx">
            <siteMapNode title="Training"
                         description="Training classes"
                         url="~/Training.aspx" />
            <siteMapNode title="Support"
                         description="Supports plans"
                         url="~/Support.aspx" />
         </siteMapNode>
    </siteMapNode>
</siteMap>
```

SiteMapPath Server Control

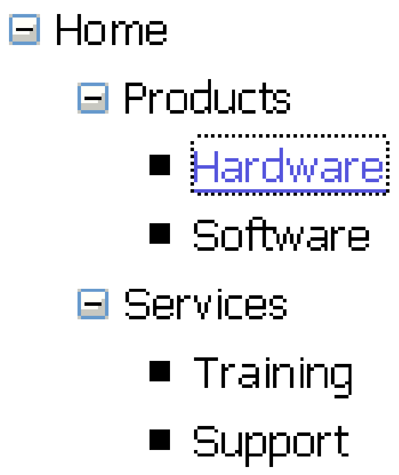
- The **`SiteMapPath`** control creates navigation functionality on the Web page


Home : Products : Hardware

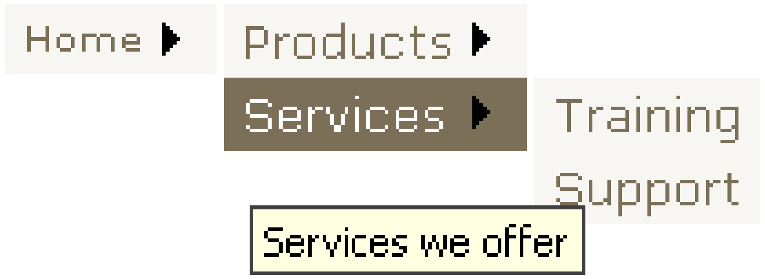
- This is a linear path defining where the end user is in the navigation structure
- Useful properties:
  - **`PathSeparator`** – any string
  - **`PathSeparatorStyle`** – allows to apply an image as a separator
  - **`PathDirection – RootToCurrent, CurrentToRoot`**
  - **`ParentLevelsDisplayed`** – number of visible parents
  - **`ShowToolTips`**

TreeView Server Control

- To present a Web site as a tree:
  1. Add a **`SiteMapDataSource`** control
  2. Add a **`TreeView`** control and set its **`DataSourceID`** property to the **`SiteMapDataSource`** object
- Features of the **`TreeView`** control:
  - Automatic data binding
  - Site navigation support
  - Highly customizable formatting
  - Programmatic access to the **`TreeView`** object model
  - Node population through client-side callbacks to the server
  - The ability to display a checkbox next to each node



Menu Server Control



- To display a site map as a menu, the **`Menu`** control must be added and its **`DataSourceID`** property must be set to an existing object of **`SiteMapDataSource`** class
- The appearance and behaviour of the **`Menu`** control is customizable
- The **`Menu`** control can be bound to the database data
- Available events:
  - **`DataBinding, DataBound, Disposed, Init, Load, MenuItemClick, MenuItemDataBound, PreRender, Unload`**

SiteMap Data Provider

- The **`SiteMapDataSource`** control is a data source to the site map data that is stored by the site map providers that are configured for the Web site
- Useful properties:
  - **`ShowStartingNode`** (**`true`** by default)
  - **`StartFromCurrentNode`** (**`false`** by default)
  - **`StartingNodeOffset`** (0 by default)
  - **`StartingNodeUrl`**

Master Pages

Master Pages

- ASP.NET master pages allow to create a consistent layout for the pages in an application
- A single master page defines the look and feel and standard behaviour that you want for all of the pages (or a group of pages) in your application
- When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page

Advantages of Master Pages

- They allow you to centralize the common functionality of your pages so that you can make updates in just one place
- They make it easy to create one set of controls and code and apply the results to a set of pages
  - For example, you can use controls on the master page to create a menu that applies to all pages
- They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered
- They provide an object model that allows you to customize the master page from individual content pages

.master Files

- A master page is an ASP.NET file with the extension **`.master`** with a predefined layout
  - It can include static texts, HTML elements, and server controls
  - The master page is identified by a special **`@Master`** directive that replaces the **`@Page`** directive

```
<%@ Master Language="C#" CodeFile="MasterPage.master.cs"
     Inherits="MasterPage" %>
```

  - The master page also contains all of the top-level HTML elements for a page, such as **`html`**, **`head`**, and **`form`**

Replaceable Content Placeholders

- The master page also includes one or more `ContentPlaceHolder` controls
  - They define regions where replaceable content will appear
  - The replaceable content is defined in content pages

```
<%@ Master Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
    1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server" >
  <title>Master page title</title>
</head>
<body>
  <form id="form1" runat="server">
    <table>
      <tr>
        <td><asp:contentplaceholder id="Main"
                  runat="server" /></td>
        <td><asp:contentplaceholder id="Footer"
                  runat="server" /></td>
      </tr>
    </table>
  </form></body></html>
```

Content Pages

- Content pages are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page

```
<% @ Page Language="C#" MasterPageFile="~/Master.master"
       Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main"
        Runat="Server">
    Main content.
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="Footer"
Runat="Server" >
    Footer content.
</asp:content>
```

Run-time Behaviour of Master Pages

1. Users request a page by typing the URL of the content page
2. When the page is fetched, the **`@Page`** directive is read
   - If the directive references a master page, the master page is read as well
   - If this is the first time the pages have been requested, both pages are compiled
3. The master page with the updated content is merged into the control tree of the content page
4. The content of individual **`Content`** controls is merged into the corresponding **`ContentPlaceHolder`** control in the master page
5. The resulting merged page is rendered to the browser

Paths to External Files

- The page created by merging the content page and the master pages is run in the context of the content page
  - The **`HttpRequest.CurrentExecutionFilePath`** property called from both the content page or the master page returns the path representing the location of the content page
- ASP.NET can modify the URLs used by the master page, if:
  - The URL is a property of a server control
  - The property is marked with the **`UrlPropertyAttribute`**
- ASP.NET cannot modify URLs on elements that are not server controls
  - It is recommended to use a server control when working with elements on master pages

Scoping Master Pages

- Content pages can be attached to a master page at three levels:
  - The page level – by using the **`MasterPageFile`** parameter of the **`@Page`** directive

```
<%@ Page Language="C#" MasterPageFile="MySite.Master %>
```

  - The application level – by making a setting in the pages element of the application's configuration file (**`Web.config`**)

```
<pages masterPageFile="MySite.Master" />
```

  - The folder level – by making the same change as for the application level but in the **`Web.config`** file located in the folder

Nested Master Pages

- Master pages can be nested, with one master page referencing another as its master
  - Nested master pages allow to create componentized master pages
- A child master page has the **`.master`** extension
  - It typically contains content controls that are mapped to content placeholders on the parent master page
  - It also has content placeholders of its own to display content supplied by its own child pages

Accessing Members of the Master Page

- To provide access to members of the master page, the **`Page`** class exposes a **`Master`** property

```
CompanyName.Text = Master.CompanyName;
```

- To access members of a specific master page from a content page, a strongly typed reference to the master page can be created by using a **`@MasterType`** directive

```
<%@ Page  masterPageFile="~/MasterPage.master"%>
<%@ MasterType  virtualPath="~/MasterPage.master"%>
```

Getting Values of Controls

- Controls contained in a **`ContentPlaceHolder`** control are not directly accessible (they are protected)
  - The **`FindControl()`** method can be used to locate specified controls on the master page

```
// Gets a reference to a TextBox inside a ContentPlaceHolder
ContentPlaceHolder mpContentPlaceHolder;
mpContentPlaceHolder = (ContentPlaceHolder)Master.FindControl(
                       "ContentPlaceHolder1");
if (mpContentPlaceHolder != null) {
    TextBox mpTextBox = (TextBox)
            mpContentPlaceHolder.FindControl("TextBox1");
    if (mpTextBox != null) {
        mpTextBox.Text = "TextBox found!";
    }
}
// Gets a reference to a Label control that is not in a
// ContentPlaceHolder control
Label mpLabel = (Label)Master.FindControl("masterPageLabel");
if (mpLabel != null) {
    Label1.Text = "Master page label = " + mpLabel.Text;
}
```

Getting Values of Controls cont.

- A public property in the master page can be created to allow to get or set the values of controls

```
public Image AnimalImage  {
    get { return this.Animal; }
    set { this.Animal = value; }
}

<asp:Image ID="Animal"
           runat="server"
           ImageUrl="~/Images/Animal.gif" />
```

```
<%@ MasterType TypeName="SiteMasterPage" %>

protected void Page_Load(object sender, EventArgs e) {
    Master.AnimalImage.ImageUrl = "~/images/procCS.gif";
}
```

Attaching Master Pages Dynamically

- A master page can be attached dynamically to a content page
  - The master page must be assigned before pages are merged
  - Typically, the **`PreInit`** event is used

```
void Page_PreInit(Object sender, EventArgs e) {
    this.MasterPageFile = "~/NewMaster.master";
}
```

Events in Master and Content Pages

1. Master page controls `Init` event
2. Content controls `Init` event
3. The master page `Init` event
4. The content page `Init` event
5. The content page `Load` event
6. The master page `Load` event
7. Content controls `Load` event
8. The content page `PreRender` event
9. The master page `PreRender` event
10. Master page controls `PreRender` event
11. Content controls `PreRender` event

Container-Specific Master Pages

- ASP.NET allows to use multiple master pages within the content page
  - Depending on the viewing container used by the end user, ASP.NET pulls the appropriate master file
  - A list of available browsers is hosted at **`C:\Windows\Microsoft.NET\Framework\v2.0____\Config\Browsers`**
    - New **`.browser`** files can be created

```
<%@ Page Language="VB"
         MasterPageFile="~/main.master"
         Mozilla:MasterPageFile="~/forMozilla.master"
         Opera:MasterPageFile="~/forOpera.master" %>
```

Themes and Skins

Themes

- A theme is a collection of property settings that allow to define the look of pages and controls
  - The look is applied consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server
- A theme is made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources
  - At a minimum, a theme will contain skins
  - When a .css file is put in the theme directory, it is applied automatically as a part of the theme

Skins

- A skin file has the **`.skin`** extension and contains property settings for individual controls such as the **`Button`**, **`Label`**, **`TextBox`**, or **`Calendar`** controls
- Control skin settings are like the control markup itself, but contain only the properties you want to set as a part of the theme
  - The default skin automatically applies to all controls of the same type

```
<asp:button runat="server"
            BackColor="lightblue" ForeColor="black" />
```

  - A named skin is a control skin with the **`SkinID`** property set
    - The named skin can be applied to a control by setting the **`SkinID`** property of the control

Scoping Themes

- Page themes
  - A page theme is a theme folder with control skins, style sheets, graphics files and other resources created in a subfolder of the **`\App_Themes`** folder in the Web site
  - Each theme has a different subfolder
- Global themes
  - A global theme is a theme that can be applied to all Web sites on a server
  - Global themes are stored in the folder named **`\Themes`** that is global to the Web server
  - **`C:\WINDOWS\Microsoft.NET\Framework\v2.0._____\ASP.NETClientFiles\Themes`**

Applying Themes

- Applying a theme to a Web site

```
<configuration>
    <system.web>
        <pages theme="ThemeName" />
    </system.web>
</configuration>
```

- Applying a theme to an individual page

```
<%@ Page Theme="ThemeName" %>
<%@ Page StyleSheetTheme="ThemeName" %>
```

- Applying a skin to a control

```
<asp:Calendar runat="server" ID="DatePicker"
    SkinID="SmallCalendar" />
```

Applying Themes Programmatically

```
Protected void Page_PreInit(object sender, EventArgs e) {
    switch (Request.QueryString["theme"]) {
        case "Blue":
            Page.Theme = "BlueTheme";
            break;
        case "Pink":
            Page.Theme = "PinkTheme";
            break;
    }
}
```

```
void Page_PreInit(object sender, EventArgs e) {
    Calendar1.SkinID = "MySkin";
}
```

Disabling Themes

- Disabling a theme for a page

```
<%@ Page EnableTheming="false" %>
```

- Disabling a theme for a control

```
<asp:Calendar id="Calendar1" runat="server"
              EnableTheming="false" />
```

The StyleSheetTheme Attribute

- The **`Page.StyleSheetTheme`** attribute works the same as the **`Theme`** attribute in that it can be used to apply a theme to a page
- The difference is visible when attributes are set locally on the page within a particular control
  - If the **`Theme`** attribute is used, the attributes are overridden
  - If the **`StyleSheetTheme`** attribute is used, the attributes are kept in place

Themes vs. CSS

- Themes can define many properties of a control or page, not just style properties
  - For example, using themes, you can specify the graphics for a **`TreeView`** control, the template layout of a **`GridView`** control, and so on
- Themes can include graphics
- Themes do not cascade the way style sheets do
  - For example, by default, property values override local property values unless you explicitly apply the theme as a style sheet theme
- Only one theme can be applied to each page – you cannot apply multiple themes to a page, unlike style sheets where multiple style sheets can be applied

Personalisation

ASP.NET Personalisation

- The ASP.NET personalisation can make an automatic association between the end user viewing the page and any data points stored for that user
- The personalisation properties that are maintained on a per-user basis are stored on the server and not on the client

Creating Personalisation Properties

1. Add the **`profile`** section to the **`Web.config`** file defining all the properties that should be stored by the personalisation engine

```
<configuration>
    <system.web>
        <profile>
            <properties>
                <add name="FirstName" />
                <add name="LastName" />
                <add name="LastVisited" />
            </properties>
        </profile>
        <authentication mode="Windows" />
    </system.web>
</configuration>
```

Creating Personalisation Properties cont.

2. When the profile is defined in the `Web.config` file, properties of the profile can be accessed from the source code

```
protected void Button1_Click(object sender, EventArgs e) {
    if (Page.User.Identity.IsAuthenticated) {
        Profile.FirstName = TextBox1.Text;
        Profile.LastName = TextBox2.Text;
        Profile.LastVisited = DateTime.Now.ToString();
        Label1.Text = "Stored information includes:<p>" +
                     "First name: " + Profile.FirstName +
                     "<br>Last name: " + Profile.LastName +
                     "<br>Last visited: " + Profile.LastVisited;
    } else {
        Label1.Text = "You must be authenticated!";
    }
}
```

```
myFirstName =
     (string)Profile.PropertyValues["FirstName"].PropertyValue;
```

Groups of Personalisation Properties

```
<profile>
    <properties>
        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <group name="MemberDetails">
            <add name="Member" />
            <add name="DateJoined" />
            <add name="PaidDuesStatus" />
        </group>
    </properties>
</profile>
```

```
Label1.Text = Profile.MemberDetails.DateJoined;
```

Defining Types of Properties

- By default, all properties are stored as type **`string`**

```
<properties>
    <add name="FirstName" type="System.String" />
    <add name="LastName" type="System.String" />
    <add name="LastVisited" type="System.DateTime" />
    <add name="Age" type="System.Integer" />
    <add name="Member" type="System.Boolean" />
</properties>
```

Using Custom Types for Properties

- **serializeAs**:
  - Binary
  - ProviderSpecific
  - String
  - XML

```
[Serializable]
public class Product {
    private string PID;
    private string CompanyProductName;

    public ShoppingCart() {}
    public string ProductID {
        get { return PID; }
        set { PID = value; }
    }
    public string ProductName {
        get { return CompanyProductName; }
        set { CompanyProductName = value; }
    }
}
```

```
<properties>
    <add name="FirstName" type="System.String" />
    <add name="LastName" type="System.String" />
    <add name="Product" type="Product" serializeAs="Binary" />
</properties>
```

Anonymous Personalisation

- By default, the anonymous personalisation is turned off because it consumes database resources

```
<configuration>
    <system.web>
        <anonymousIdentification enabled="True" />
    </system.web>
</configuration>
```

- For an anonymous user, information is stored by default as a cookie on the end user's machine
  - Additional information (the personalisation properties that you enable for anonymous users) is stored in the specified data store on the server

Settings of Anonymous Personalisation

- Behaviour of the anonymous personalisation can be set up using attributes of the **`anonymousIdentification`** section in the configuration file
  - The name of the cookie - **`cookieName`**
  - The length of time the cookie is stored - **`cookieTimeout`**
  - How the identifiers should be stored - **`cookieless`**
    - **`UseUri`** – no cookie, id in the URL of the page
    - **`UseCookies`** – using cookies (the default value)
    - **`AutoDetect`**
    - **`UseDeviceProfile`** – depending of the **`HttpBrowserCapabilities`** setting

Using Anonymous Personalisation

- An identifier of the user:
  - `Request.AnonymousId`
- Events:
  - `AnonymousIdentification_OnCreate`
  - `AnonymousIdentification_OnRemove`
- Choosing properties of the anonymous personalisation:

```
<properties>
    <add name="FirstName" type="System.String" />
    <add name="LastName" type="System.String" />
    <add name="LastVisited" type="System.DateTime"
                                  allowAnonymous="true" />
    <add name="Age" type="System.Integer" />
    <add name="Member" type="System.Boolean" />
</properties>
```

Personalisation Providers

- The middle tier of the personalisation model, the personalisation API layer, communicates with a series of default data providers
  - By default, Microsoft SQL Server Express Edition is used

```
<configuration>
    <connectionStrings>
        <clear />
        <add name="LocalSqlServer"
             connectionString="data source=.\SQLEXPRESS;
                   Integrated Security=SSPI;
                   AttachDBFilename=|DataDirectory|aspnetdb.mdf;
                   User Instance=true"
             providerName="System.Data.SqlClient" />
    </connectionStrings>
</configuration>
```

  - SQL Server 7.0 or later can be used

ASP.NET SQL Server Setup Wizard

- The ASP.NET SQL Server Setup Wizard is an easy-to-use tool that facilitates setup of the SQL Server to work with the personalisation framework
- **`aspnet_regsql.exe`** - there are two ways to set up the database:
  - Command-Line Tool
  - GUI Tools
- The **`aspnetdb`** database is created by the wizard

Using Multiple Providers

```
<configuration>
    <system.web>
        <profile defaultProvider="AspNetSqlProvider">
            <properties>
                <add name="FirstName" />
                <add name="LastName" />
                <add name="LastVisited" />
                <add name="Age" />
                <add name="Member"
                     provider="AspNetSql2000Provider" />
            </properties>
        </profile>
    </system.web>
</configuration>
```

Membership and Role Management

Authentication and Authorization

- The authentication
  - Helps to verify that the user is, in fact, who the user claims to be
  - The application obtains credentials from a user and validates those credentials against some authority
- The authorization
  - Limits access rights by granting or denying specific permissions to an authenticated identity

Authentication Providers

- There are 3 authentication providers built into ASP.NET
  - Windows Authentication Provider
    - Information on how to use Windows authentication in conjunction with IIS authentication
  - Forms Authentication Provider
    - Information on how to create an application-specific login form and perform authentication using your own code
  - Passport Authentication Provider
    - Information about the centralized authentication service provided by Microsoft

Windows Authentication Provider

- The Windows Authentication treats the user identity supplied by Microsoft Internet Information Services (IIS) as the authenticated user in an ASP.NET application
- It is the default authentication mechanism for ASP.NET applications

```
<system.web>
    <authentication mode="Windows"/>
</system.web>
```

- By default, the identity of the ASP.NET worker process is used by the operating system to check all permissions (e.g. file or database access using integrated security)
  - It can be changed by enabling impersonation and setting the `User` property

Types of Windows Authentication

- Basic – the simplest and least secure
  - The browser presents a standard Windows-supplied dialog box for the user to enter his credentials (a username and password)
  - The username and password are sent to the server encoded as a Base64 string (in clear text)
- Digest
  - Similar to basic authentication, but the credentials are encrypted and a hash is sent over the network to the server
  - Works only with IE 5.x or higher and Windows 2000 or higher
- Integrated Windows authentication
  - Uses the current users' credentials presented at the time they logged into Windows
  - A dialog box is never presented to the user to gather credentials unless the Windows logon credentials are inadequate for a requested resource

Forms Authentication Provider

- The Forms Authentication allows to authenticate users using your own code and then maintain an authentication token in a cookie or in the page URL
- The **`authentication`** configuration element is used to configure forms authentication
- ASP.NET membership provides a way to store and manage user information and includes methods to authenticate users
- ASP.NET login controls work with ASP.NET membership and allow to prompt users for credentials, validate users, recover or replace passwords, and so on

How to Use Forms Authentication

- In the **`Web.config`** file:
  1. Set the authentication mode
  2. Set the login form and the name of the cookie that contains the authentication ticket
  3. Deny access for anonymous users

```
<system.web>
    <authentication mode="Forms">
        <forms loginUrl="logon.aspx" name=".ASPXFORMSAUTH">
        </forms>
    </authentication>
    <authorization>
        <deny users="?" />
    </authorization>
</system.web>
```

How to Use Forms Authentication cont.

- Create the logon page:
  1. Add controls to allow the user specify his data (in most cases a login and password)
  2. Write code to verify the data

```
void Logon_Click(object sender, EventArgs e) {
    if ((UserEmail.Text == "jchen@contoso.com") &&
        (UserPass.Text == "37Yj*99Ps"))
    {
        FormsAuthentication.RedirectFromLoginPage(
                         UserEmail.Text, Persist.Checked);
    } else {
        Msg.Text = "Invalid credentials. Please try again.";
    }
}
```

How to Use Forms Authentication cont.

- Access to all pages of the application will be granted only for users successfully authenticated on the logon page
  - The login is available in the `User` property

```
void Page_Load(object sender, EventArgs e) {
    Welcome.Text = "Hello, " + Context.User.Identity.Name;
}

void Signout_Click(object sender, EventArgs e) {
    FormsAuthentication.SignOut();
    Response.Redirect("Logon.aspx");
}
```

Forms Authentication Control Flow

```
1.  GET /default.aspx
2.  302 Found
    Location: http://abc.com/logon.aspx?RETURNURL=/default.aspx
3.  GET /logon.aspx?RETURNURL=/default.aspx
4.  200 OK
5.  POST /logon.aspx?RETURNURL=/default.aspx
6.  302 Found
    Location: /default.aspx
7.  GET /default.aspx
8.  200 OK
    Set-Cookie: ASPXTICKET=ABCDEFG12345;Path=/
```

Future requests by the same browser session will be authenticated when the module inspects the cookie

It is possible to create a persistent cookie that can be used for future sessions, but only until the cookie's expiration date

Forms Authentication Credentials

- Forms authentication credentials that are used to validate users at logon can be stored in an external data source or in the application configuration file
  - The **`Authenticate()`** method can be used to compare the credentials collected from the user to the list of user/password pairs in the **`credentials`** section of the **`Web.config`** file
  - Passwords can be stored using:
    - Clear text (**`Clear`**)
    - The MD5 hash digest (**`MD5`**) – better performance than SHA1
    - The SHA1 has digest (**`SHA1`**) – improved security

```
<credentials passwordFormat="SHA1" >
    <user name="Kim"
          password="07B7F3EE06F278DB966BE960E7CBBD103DF30CA6"/>
    <user name="John"
          password="BA56E5E0366D003E98EA1C7F04ABF8FCB3753889"/>
</credentials>
```

FormsAuthentication Class

- `Initialize()` – reads configuration settings and gets encryption values for the application
- `Authenticate()` – attempts to validate the credentials
- `Encrypt()`, `Decrypt()` – a `FormsAuthenticationTicket` to string and vice versa containing an encrypted authentication ticket suitable for an HTTP cookie
- `GetRedirectUrl()` – returns the redirection URL for the request
- `RedirectFromLoginPage()` – redirects an authenticated user to the originally requested URL
- `HasPasswordForStoringInConfigFile()` – produces a hash for a password
- `RenewTicketIfOld()` – updates the sliding expiration
- `SetAuthCookie()` – creates an authentication ticket and attaches it to the cookie collection of the response
- `SignOut()` – removes the authentication ticket by setting the authentication cookie or URL text to an empty value

FormsAuthentication Class cont.

- **`FormsCookieName, FormsCookiePath`** – gets the cookie name, path for the application
- **`CookiesSupported`** – whether the application is configured to support cookieless forms authentication
- **`CookieMode`** – whether the application is configured for cookieless forms authentication
- **`CookieDomain`** – the domain of the forms authentication cookie
- **`DefaultUrl, LoginUrl`** – URLs of the default and login addresses
- **`RequireSSL`** – whether cookies must be transmitted using SSL
- **`SlidingExpiration`** – whether sliding expiration is enabled
- **`EnableCrossAppRedirects`** – whether authenticated users can be redirected to URLs in other Web applications when the forms authentication ticket is not stored in a cookie

- **`FormsAuthentication_OnAuthenticate`** - in the **`Global.asax`** file

ASP.NET Authorization

- The authorization determines whether an identity should be granted access to a specific resource
- There are two ways to authorize access:
  - The file authorization – it determines whether a user should have access to the file
  - The URL authorization – can be used to selectively allow or deny access to arbitrary parts of an application

Using URL Authorization

- The **`authorization`** section of the **`Web.config`** file is used to configure the URL authorization
  - **`allow, deny`**
  - *users* – names of users, "**`?`**" for anonymous users, "**`*`**" for all authenticated users
  - *roles* – identifies a role (a **`RolePrincipal`** object)
  - *verbs* – defines the HTTP verbs to which the action applies (**`GET, HEAD, POST, *`**)

```
<authorization>
  <allow verbs="GET" users="*"/>
  <allow verbs="POST" users="Kim"/>
  <deny verbs="POST" users="*"/>
</authorization>
```

```
<authorization>
    <allow users="John"/>
    <deny users="*"/>
/authorization>
```

```
    <allow users="Kim"/>
    <allow roles="Admins"/>
    <deny users="John"/>
    <deny users="?"/>
</authorization>
```

ASP.NET Impersonation

- When using the impersonation, ASP.NET applications can be executed with the Windows identity (user account) of the user making the request

```
<configuration>
    <system.web>
        <identity impersonate="true"/>
    </system.web>
</configuration>
```

```
<identity impersonate="true"
          userName="contoso\Jane"
          password="E@1bp4!T2" />
```

```
String username =
  System.Security.Principal.WindowsIdentity.GetCurrent().Name;
```

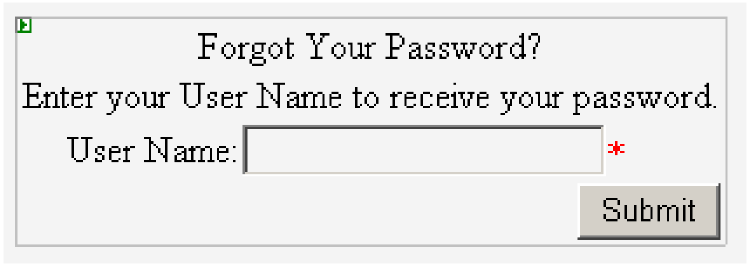
ASP.NET Membership

- Features of the ASP.NET membership:
  - Creating new users and passwords
  - Storing membership information (user names, passwords, and supporting data) in Microsoft SQL Server, Active Directory, or other
  - Authenticating users who visit the site
    - ASP.NET login controls can be used to create a complete authentication system that requires little or no code
  - Managing passwords (creating, changing, and resetting)
  - Exposing a unique identification for authenticated users
    - Can be used in the application
    - Integrates with the ASP.NET personalisation and role-management (authorization) systems
  - A custom membership provider can be created

Login Controls

- `Login` - displays a user interface for the user authentication
- `LoginView` - allows to display different information to anonymous and logged-in users (using two templates)
- `LoginStatus` - displays a login link for users who are not authenticated and a logout link for authenticated users
- `LoginName` - displays a user's login name (works also for Windows authentication)
- `PasswordRecovery` - allows a user's password to be retrieved based on the e-mail address
- `CreateUserWizard` - allows users to change their password

Login Controls cont.

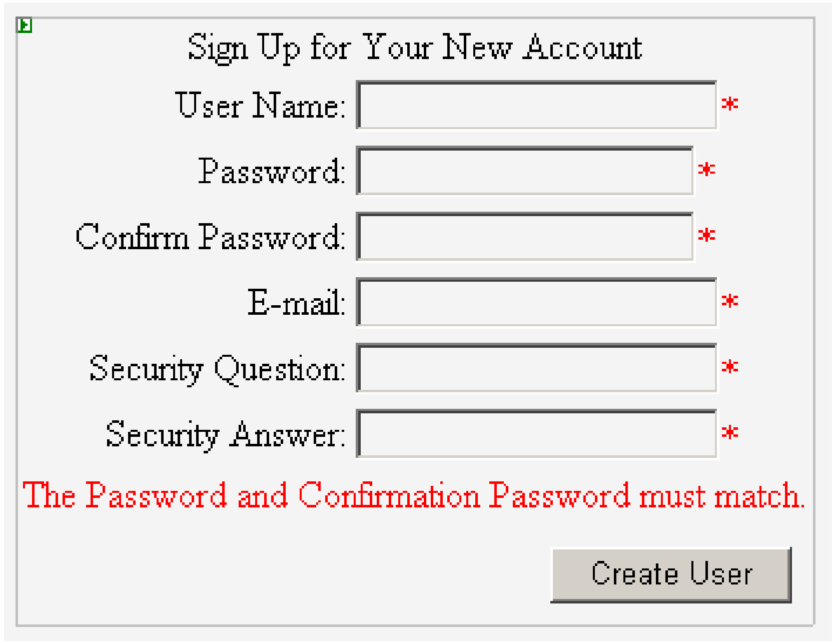


Forgot Your Password?

Enter your User Name to receive your password.

User Name: *

Submit



Sign Up for Your New Account

User Name: *

Password: *

Confirm Password: *

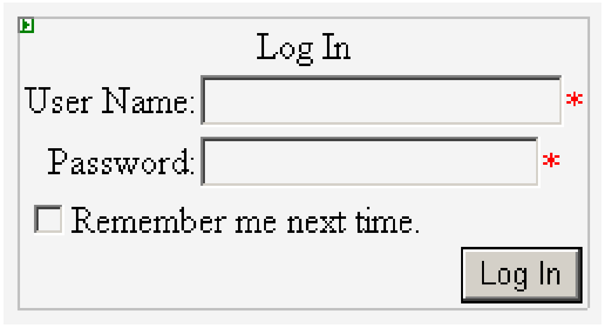
E-mail: *

Security Question: *

Security Answer: *

The Password and Confirmation Password must match.

Create User



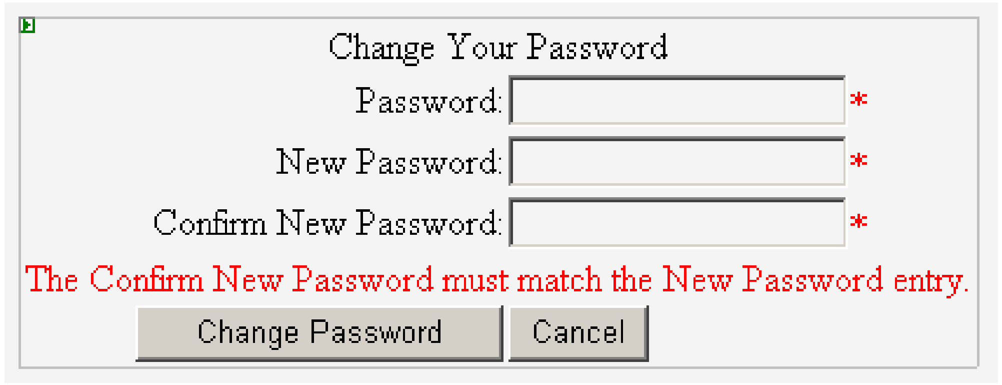
Log In

User Name: *

Password: *

☐ Remember me next time.

Log In



Change Your Password

Password: *

New Password: *

Confirm New Password: *

The Confirm New Password must match the New Password entry.

Change Password Cancel

Role Management

- The role management allows to treat groups of users as a set of units by assigning users to roles such as manager, sales, member, and so on
- Ways of roles management:
  - The ASP.NET Web Site Administration Tool

```
<roleManager enabled="true" cacheRolesInCookie="true" >
</roleManager>
```

  - Programmatically

```
Roles.CreateRole("members");
```

```
Roles.AddUsersToRole("JoeWorden", "manager");
string[] userGroup = new string[2];
userGroup[0] = "JillShrader";
userGroup[1] = "ShaiBassli";
Roles.AddUsersToRole(userGroup, "members");
```

Working with Roles

- Information about the logged-in user is available to the application from the **`User`** property
- When roles are enabled, ASP.NET looks up the roles for the current user and adds them to the **`User`** object

```
if (User.IsInRole("members")) {
    buttonMembersArea.Visible = True;
}
```

```
string[] userRoles = ((RolePrincipal)User).GetRoles();
```

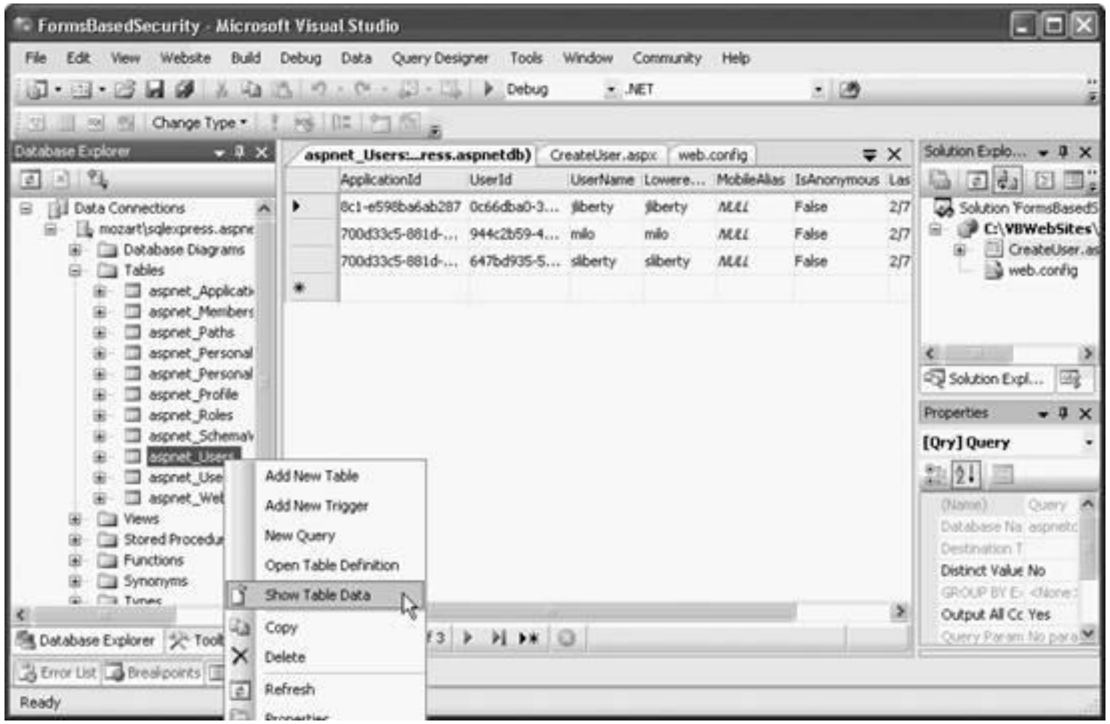
- Based on user's roles, the **`LoginView`** control can dynamically create an interface for the user

Role Management Providers

- Role management services use the provider model to separate the functionality of role management from the data store that contains role information
- The .NET Framework includes the following providers that maintain role information in different data stores:
  - SQL Server (the default provider)
  - Windows – useful only if the application runs on a network where all users have domain accounts
  - Authorization Manager (AzMan) – an XML file or a directory-based policy store
- It is possible to create custom role providers

SQL Server Role Management Provider

- It is the default provider, it can be used with SQL Server or SQL Server Express
- The **`aspnet_regsql.exe`** tool creates the required database and all its tables
  - The database server instance is **`.\SQLEXPRESS`**
  - The authentication type is **`Windows`**
  - The name of the database is **`aspnetdb`**



ApplicationId	UserId	UserName	Lowere...	MobileAlias	IsAnonymous	Las
8c1-e598ba6ab287	0c66dba0-3...	jliberty	jliberty	NULL	False	2/7
700d33c5-881d-...	944c2b59-4...	milo	milo	NULL	False	2/7
700d33c5-881d-...	647bd935-5...	sliberty	sliberty	NULL	False	2/7

State Management

ASP.NET State Management

- The HTTP protocol is stateless
- To keep information between requests, one of the following ways can be used:
  - Client-based options:
    - View state
    - Control state
    - Hidden fields
    - Cookies
    - Query strings
  - Server-based options:
    - Application state
    - Session state
    - Profile properties

View State

- A Web application is stateless
  - A new instance of the Web page class is created each time the page is requested from the server
- View state is a way to store information directly on the page that should persist between postbacks
  - When the page is posted back to the server, the contents of view state are sent as a part of the page postback information
- View state data is stored in one or more hidden fields
  - View state information can be accessed using the page's `ViewState` property
  - Since view state is sent as a hidden field, changes to view state can be made until the `PreRenderComplete` event

Considerations for Using View State

- View state information is serialized into XML and then encoded using base64 encoding
  - If view state contains a large amount of information, it can affect performance of the page
  - If the amount of data in a hidden field becomes large, some proxies and firewalls will prevent access to the page that contains them
- Some mobile devices do not allow hidden fields at all

Example of Saving Values in View State

```
ArrayList PageArrayList;

ArrayList CreateArray() {
    ArrayList result = new ArrayList(4);
    result.Add("item 1");
    result.Add("item 2");
    result.Add("item 3");
    result.Add("item 4");
    return result;
}

void Page_Load(object sender, EventArgs e) {
    if (ViewState["arrayListInViewState"] != null) {
        PageArrayList = (ArrayList)ViewState["myArrayList"];
    } else {
        // ArrayList isn't in the view state
        PageArrayList = CreateArray();
    }
    // Code that uses PageArrayList.
  }

void Page_PreRender(object sender, EventArgs e) {
    // Save PageArrayList before the page is rendered.
    ViewState.Add("arrayListInViewState", PageArrayList);
}
```

Pros and Cons of View State

- Advantages
  - No server resources are required
  - Simple implementation (it does not require any custom programming to use)
  - Enhanced security features (the values in view state are hashed, compressed, encoded for Unicode implementations, and optionally encrypted)
- Disadvantages
  - Performance considerations (storing large values can cause the page to slow down)
  - Device limitations (mobile devices might not have the memory capacity to store a large amount of view-state data)

Control State

- In addition to view state, ASP.NET supports a page-state feature called control state
- The page uses control state to persist control information that must be retained between postbacks
  - It works even if the view state is disabled for the page or for a control
- Like view state, control state is stored in one or more hidden fields
- An example of using the control state:
  - A custom control which needs to store some data between requests

Pros and Cons of Control State

- Advantages
  - No server resources are required (by default, control state is stored in hidden fields on the page)
  - Reliability (control state works even when the view state is turned off)
  - Versatility (custom adapters can be written to control how and where control-state data is stored)
- Disadvantages
  - Some programming is required (it is a custom state-persistent mechanism, the developer has to write code to save and load control state)

Pros and Cons of Hidden Fields

- Advantages
  - No server resources are required
  - Widespread support
  - Simple implementation
- Disadvantages
  - Potential security risks (the content can be manually encrypted and decrypted, but it requires extra coding and overhead)
  - Simple storage architecture (just simple string values)
  - Performance considerations (storing large values can cause the page to slow down)
  - Storage limitations (if the amount of data in a hidden field becomes very large, some proxies and firewalls will prevent access to the page that contains them)

Cookies

- A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser
  - The cookie contains information the Web application can read whenever the user visits the site
- Cookie limitations:
  - Most browsers support cookies of up to 4096 bytes
  - Most browsers allow only 20 cookies per site
    - Some browsers also put an absolute limit of the number of cookies they will accept from all sites combined
  - The user can set his browser to refuse cookies

Writing Cookies

```
Response.Cookies["userName"].Value = "patrick";
Response.Cookies["userName"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("lastVisit");
aCookie.Value = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

```
Response.Cookies["userInfo"]["userName"] = "patrick";
Response.Cookies["userInfo"]["lastVisit"] = DateTime.Now.ToString();
Response.Cookies["userInfo"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = new HttpCookie("userInfo");
aCookie.Values["userName"] = "patrick";
aCookie.Values["lastVisit"] = DateTime.Now.ToString();
aCookie.Expires = DateTime.Now.AddDays(1);
Response.Cookies.Add(aCookie);
```

Controlling Cookies

- Limiting a cookie to a folder or application:

```
HttpCookie appCookie = new HttpCookie("AppCookie");
appCookie.Value = "written " + DateTime.Now.ToString();
appCookie.Expires = DateTime.Now.AddDays(1);
appCookie.Path = "/Application1";
Response.Cookies.Add(appCookie);
```

- Limiting a cookie domain space:

```
Response.Cookies["domain"].Value = DateTime.Now.ToString();
Response.Cookies["domain"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["domain"].Domain = "support.contoso.com";
```

Reading Cookies

```
if (Request.Cookies["userName"] != null) {
    Label1.Text = Server.HtmlEncode(
                             Request.Cookies["userName"].Value);
}
if (Request.Cookies["userName"] != null) {
    HttpCookie aCookie = Request.Cookies["userName"];
    Label1.Text = Server.HtmlEncode(aCookie.Value);
}
```

```
if (Request.Cookies["userInfo"] != null) {
    Label1.Text = Server.HtmlEncode(
                      Request.Cookies["userInfo"]["userName"]);
    Label2.Text = Server.HtmlEncode(
                      Request.Cookies["userInfo"]["lastVisit"]);
}
```

- Using the **`HtmlEncode()`** method makes certain that a malicious user has not added executable script into the cookie

Reading Cookie Collections

```
for (int i = 0; i < Request.Cookies.Count; i++) {
  aCookie = Request.Cookies[i];
  output.Append("Name = " + aCookie.Name + "<br />");
  if (aCookie.HasKeys) {
    for (int j = 0; j < aCookie.Values.Count; j++) {
      subkeyName = Server.HtmlEncode(aCookie.Values.AllKeys[j]);
      subkeyValue = Server.HtmlEncode(aCookie.Values[j]);
      output.Append("Subkey name = " + subkeyName + "<br />");
      output.Append("Subkey value = " + subkeyValue + "<br/>");
    }
  } else {
    output.Append("Value = " + Server.HtmlEncode(aCookie.Value) +
                  "<br /><br />");
  }
}
Label1.Text = output.ToString();
```

Modifying and Deleting Cookies

- Modifying a value of a cookie:

```
int counter;
if (Request.Cookies["counter"] == null) {
    counter = 0;
} else {
    counter = int.Parse(Request.Cookies["counter"].Value);
}
counter++;
Response.Cookies["counter"].Value = counter.ToString();
Response.Cookies["counter"].Expires = DateTime.Now.AddDays(1);
```

- Deleting a cookie:

```
if (Request.Cookies["counter"] != null) {
    HttpCookie aCookie = new HttpCookie("counter");
    aCookie.Expires = DateTime.Now.AddDays(-1);
    Response.Cookies.Add(aCookie);
}
```

Testing whether Cookies are Accepted

```
protected void Page_Load(object sender, EventArgs e) {
  if (!Page.IsPostBack) {
    if (Request.QueryString["AcceptsCookies"] == null) {
      Response.Cookies["TestCookie"].Value = "ok";
      Response.Cookies["TestCookie"].Expires =
                                        DateTime.Now.AddMinutes(15);
      Response.Redirect("TestForCookies.aspx?redirect=" +
                        Server.UrlEncode(Request.Url.ToString()));
    } else {
      Label1.Text = "Accept cookies = " +
        Server.UrlEncode(Request.QueryString["AcceptsCookies"]);
    }
```

```
protected void Page_Load(object sender, EventArgs e) {
  string redirect = Request.QueryString["redirect"];
  string accepts = "no";
  if (Request.Cookies["TestCookie"] != null) {
    accepts = "yes";
    Response.Cookies["TestCookie"].Expires =
            DateTime.Now.AddDays(-1);  // delete the cookie
  }
  Response.Redirect(Request.QueryString["redirect"] +
                 "?AcceptsCookies=" + accepts, true);
}
```

Pros and Cons of Cookies

- Advantages
  - Configurable expiration rules
  - No server resources are required
  - Simplicity
  - Data persistence
- Disadvantages
  - Size limitations
  - User-configured refusal
  - Potential security risks (users can manipulate cookies on their computer; hackers have historically found ways to access cookies from other domains on a user's computer)

Pros and Cons of Query Strings

- Advantages
  - No server resources are required
  - Widespread support
  - Simple implementation
- Disadvantages
  - Potential security risks (information in the query string is directly visible to the user via the browser's user interface)
  - Limited capacity (a 2083-character limit on the length of URLs)

Application State

- Application state is a data repository available to all classes in an ASP.NET application
- It is stored in memory on the server and is faster than storing and retrieving information from a database
- It applies to all users and sessions
- Application state is stored in an instance of the **`HttpApplicationState`** class
  - This class exposes a key-value dictionary of objects
  - It is most often accessed through the **`Application`** property of the **`HttpContext`** class

Application State Considerations

- Resources – storing large blocks of data in application state can fill up server memory, causing the server to page memory to disk
- Volatility – the data is lost when the application is stopped or restarted
- Scalability – application state is not shared among multiple servers serving the same application, as in a Web farm
- Concurrency – application state can be accessed simultaneously by many threads
  - The **`Lock()`** and **`UnLock()`** methods can be used to ensure data integrity

Using Application State

```
Application["Message"] = "Welcome to the Contoso site.";
Application["PageRequestCount"] = 0;
```

```
Application.Lock();
Application["PageRequestCount"] =
    ((int)Application["PageRequestCount"])+1;
Application.UnLock();
```

```
if (Application["AppStartTime"] != null) {
    DateTime myAppStartTime = (DateTime)Application["AppStartTime"];
}
```

Pros and Cons of Application State

- Advantages
  - Simple implementation
  - Application scope
- Disadvantages
  - Application scope
  - Limited durability of data
  - Resource requirements
- Careful design and implementation of application state can increase Web application performance
  - For example, placing a commonly used, relatively static dataset in application state can increase site performance

Session State

- ASP.NET session state allows to store and retrieve values for a user
- It identifies requests from the same browser during a limited time window as a session
- Session variables are stored in a **`SessionStateItemCollection`** that is exposed through the **`HttpContext.Session`** property
- Sessions are identified by a unique session identifier stored in the **`SessionID`** property
  - **`SessionID`** values are stored in a cookie by default
  - The application can be configured to store **`SessionID`** values in the URL for a "cookieless" session

Session State Modes

- The available session state modes:
  - InProc – session state is stored in memory on the Web server (the default)
  - StateServer – using a separate process
    - Session state is preserved if the application is restarted
    - Session state is available to multiple Web servers
  - SQLServer – using a SQL Server database
  - Custom – possibility to specify a custom storage provider
  - Off – disables session state

```
<configuration>
  <system.web>
    <sessionState mode="StateServer"
      stateConnectionString="tcpip=SampleStateServer:42424"
      cookieless="false"
      timeout="20"/>
  </system.web>
</configuration>
```

Using Session State

```
string firstName = "Jeff";
string lastName = "Smith";
string city = "Seattle";
Session["FirstName"] = firstName;
Session["LastName"] = lastName;
Session["City"] = city;
```

```
string firstName = (string)(Session["First"]);
string lastName = (string)(Session["Last"]);
string city = (string)(Session["City"]);
```

```
if (Session["City"] == null) {
   // No such value in the session state; take an appropriate action
}
```

Pros and Cons of Session State

- Advantages
  - Simple implementation
  - Session-specific events
  - Data persistence
  - Platform scalability
  - Cookieless support
  - Extensibility (custom providers can be created)
- Disadvantages
  - Performance considerations

Profile Properties

- Profile properties allow to store user-specific data
- Unlike session state, the profile data is not lost when a user's session expires
- The ASP.NET profile allows to easily manage user information without requiring to create and maintain custom database
  - It makes the user information available using a strongly typed API
  - Objects of any type can be stored in the profile

Pros and Cons of Profile Properties

- Advantages
  - Data persistence
  - Platform scalability
  - Extensibility
- Disadvantages
  - Performance considerations
  - Additional configuration requirements (the profile providers must be configured and all of the profile properties must be pre-configured)
  - Data maintenance (the application must call the appropriate cleanup mechanisms)

HttpContext.Current.Items

- The **`Items`** collection of **`HttpContext`** is an **`IDictionary`** key/value collection of objects shared across the life of a single **`HttpRequest`**
- It allows to:
  - Share content between **`IHttpModules`** and **`IHttpHandlers`**
  - Communicate between two instances of the same **`UserControl`** on the same page
  - Store the results of expensive calls that might otherwise happen twice or more on a page

User Controls
Custom Controls
HTTP Modules
HTTP Handlers

User Controls

- User controls are containers into which markup and Web server controls can be put
  - The user control can be treated as a unit, properties and methods can be defined for it
- A user control is created in the same way as an ASP.NET page
  - The file extension is .ascx
  - Instead of an **`@Page`** directive, it contains an **`@Control`** directive
  - A user control does not have the **`html`**, **`body`**, or **`form`** elements
- A user control is added to a page by registering it on the host page, using the **`@Register`** directive

```
<%@ Register src="Spinner.ascx" tagname="Spinner" tagprefix="uc1" %>
```

Creating Instances Programmatically

- In the user control:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="Spinner.ascx.cs" Inherits="WebApplication2.Spinner" %>
```

- In the target page:

```
<%@ Reference Control="Spinner.ascx" %>
```

- In code of the target page:

```
private Spinner Spinner1;

protected void Page_Load(object sender, EventArgs e) {
    Spinner1 = (Spinner)LoadControl("Spinner.ascx");
    Spinner1.MaxValue = 20;
    Spinner1.MinValue = 10;
    PlaceHolder1.Controls.Add(Spinner1);
}

protected void Button1_Click(object sender, EventArgs e) {
    Label1.Text = Spinner1.CurrentNumber.ToString();
}
```

Custom Controls

- Custom controls can be derived from:
  - **`System.Web.UI.Control`**
    - The basic functionality required to participate in the Page framework
    - It implements **`IComponent`** (so it works with the Visual Studio Toolbox, Property Browser etc.)
  - **`System.Web.UI.WebControls.WebControl`**
    - The common functionality to create controls that render a visual HTML representation
    - Support for many basic styling elements such as **`Font`**, **`Width`**, and **`Height`**
  - Any existing control

```
namespace Samples.AspNet.CS.Controls {
  [AspNetHostingPermission(SecurityAction.Demand,
        Level = AspNetHostingPermissionLevel.Minimal),
  AspNetHostingPermission(SecurityAction.InheritanceDemand,
        Level=AspNetHostingPermissionLevel.Minimal),
  DefaultProperty("Email"),
  ParseChildren(true, "Text"),
  ToolboxData("<{0}:MailLink runat=\"server\"> </{0}:MailLink>")]
  public class MailLink : WebControl {

    [Bindable(true), Category("Appearance"),
     DefaultValue(""), Description("The e-mail address.")]
    public virtual string Email {
      get {
        string s = (string)ViewState["Email"];
        return (s == null) ? String.Empty : s;
      } set {
        ViewState["Email"] = value;
      }
    }

    protected override HtmlTextWriterTag TagKey {
      get { return HtmlTextWriterTag.A; }
    }
```

```
[Bindable(true), Category("Appearance"), DefaultValue(""),
 Description("The text to display on the link."), Localizable(true),
 PersistenceMode(PersistenceMode.InnerDefaultProperty)]
 public virtual string Text {
   get {
     string s = (string)ViewState["Text"];
     return (s == null) ? String.Empty : s;
   }
   set { ViewState["Text"] = value; }
 }

 protected override void AddAttributesToRender(
                                        HtmlTextWriter writer) {
   base.AddAttributesToRender(writer);
   writer.AddAttribute(HtmlTextWriterAttribute.Href,
       "mailto:" + Email);
 }

 protected override void RenderContents(HtmlTextWriter writer) {
   if (Text == String.Empty) {
     Text = Email;
   }
   writer.WriteEncodedText(Text);
 }
```

Common Attributes for Custom Controls

- **`Bindable`** – **`true`** to display the control in the data bindings dialog box
- **`Browsable`** – **`true`** to be displayed in the designer
- **`Category`** – in which category the control will be displayed when the Properties dialog is sorted by category
- **`DefaultProperty`**
- **`DefaultValue`**
- **`Description`** – visible in the description box in the Properties panel
- **`ToolboxData`** – used by Visual Studio to provide the tag when the object is dragged from the toolbox

Example of Derived Control

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:CountedButton runat=server>" +
             "</{0}:CountedButton>")]
public class CountedButton : System.Web.UI.WebControls.Button
{
    // constructor initializes view state value public
    CountedButton() {
        this.Text = "Click me";
        ViewState["Count"] = 0;
    }
    // count as property maintained in view state
    public int Count {
        get { return (int)ViewState["Count"]; }
        set { ViewState["Count"] = value; }
    }
    // override the OnClick to increment the count,
    // update the button text and then invoke the base method
    protected override void OnClick(EventArgs e) {
        ViewState["Count"] = ((int)ViewState["Count"]) + 1;
        this.Text = ViewState["Count"] + " clicks";
        base.OnClick(e);
    }
}
```

HTTP Modules

- An HTTP module is an assembly that is called on every request made to the application
- HTTP modules are called as part of the ASP.NET request pipeline and have access to life cycle events throughout the request
- The HTTP module must implement the **`IHttpModule`**
  - In the **`Init()`** method interesting application events can be subscribed
  - The **`Dispose()`** method is used to free resources
- Custom HTTP module is registered in the application's **`Web.config`** file
  - An instance of the module is created and the **`Init()`** method is called when the application object is created
  - When subscribed event is raised, the appropriate method in the module is called

Events Available for HTTP Modules

- `BeginRequest` – a new HTTP request
- `AuthenticateRequest` – authenticating the user
- `AuthorizeRequest` – authorizing the user
- `AcquireRequestState` – acquiring of the Session State
- `PreRequestHandlerExecute` – just before executing of the handler
- `PostRequestHandlerExecute` – just after executing the handler
- `PreSendRequestHeaders` – just before sending headers to the client
- `PreSendRequestContent` – just before sending contents to the client
- `EndRequest` – just before sending the response to the client
- `Error` – an unhandled exception

Applications of HTTP Modules

- Sample applications of HTTP modules:
  - Custom authentication or other security checks before the requested page, XML Web service, or handler is called
  - Statistics and logging
  - Custom headers of footers
- The HTTP module can be added to the GAC and registered in the **`Machine.config`** file, so it can be reused across applications

Creating HTTP Module

1. Create a class that implements the **`IHttpModule`**
2. Write a handler for the **`Init()`** method
   - Initialize the module
   - Subscribe to application events
3. Write code for the events
4. Optionally implement the **`Dispose()`** method
5. Register the module in the **`Web.config`** file

```
<configuration>
    <system.web>
        <httpModules>
           <add name="HelloWorldModule" type="HelloWorldModule"/>
        </httpModules>
    </system.web>
</configuration>
```

Sample HTTP Module

```
using System.Web;

public class AppendMessage : IHttpModule
{
    private HttpContext current = null;

    public void Init(System.Web.HttpApplication context)
    {
        current = context.Context;
        context.PreSendRequestContent +=
                new EventHandler(context_PreSendRequestContent);
    }

    void context_PreSendRequestContent(object sender, EventArgs e)
    {
        // alter the outgoing content by adding a HTML comment
        string message = "<!-- Processed at " +
                         System.DateTime.Now.ToString() +
                         " by a custom HTTP module -->";
        HttpApplication application = (HttpApplication)sender;
        application.Context.Response.Write(message);
    }
}
```

HTTP Handlers

- An ASP.NET HTTP handler is the process that runs in response to a request made to an ASP.NET Web application
- ASP.NET maps HTTP requests to HTTP handlers based on a file name extension
  - Each HTTP handler enables processing of individual HTTP URLs or groups of URL extensions within an application
- Sample built-in HTTP handlers in ASP.NET:
  - ASP.NET Page Handler (*.aspx)
  - Web service handler (*.asmx)
  - ASP.NET user control handler (*.ascx)
  - Trace handler (trace.axd)
- An HTTP handler can also be requested directly for serving some special content, e.g. images or files
  - .ashx extension is usually used for such handlers

Creating HTTP Handlers

- To create a custom HTTP handler, create a class that implements the **`IHttpHandler`** interface (or the **`IHttpAsyncHandler`** for an asynchronous handler)
  - Implement the **`IsReusable`** property and the **`ProcessRequest()`** method
- Asynchronous HTTP handlers allow to start an external process and continue the processing of the handler without waiting for the external process to finish
  - The **`BeginProcessRequest()`** and **`EndProcessRequest()`** method must be implemented

Sample HTTP Handler

```
using System.Web;
public class HelloWorldHandler : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        HttpRequest Request = context.Request;
        HttpResponse Response = context.Response;
        Response.Write("<html>");
        Response.Write("<body>");
        Response.Write("<h1>A custom HTTP handler.</h1>");
        Response.Write("</body>");
        Response.Write("</html>");
    }

    public bool IsReusable
    {
        // return true to
        // enable pooling
        get { return false; }
    }
}
```

```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*" path="*.sample"
           type="HelloWorldHandler"/>
    </httpHandlers>
  </system.web>
</configuration>
```

Debugging, Error Handling, and Health Monitoring

ASP.NET Debugging

- To enable debugging:
  - For all pages of the application: use the **`compilation`** section of the **`Web.config`** file
  - For single pages: add **`debug=true`** to the **`@Page`** directive
- The debug mode should be turned off before deploying
  - The application compiled into a debug build performs considerably more slowly
  - In the debug mode, more information is exposed in the stack when an error occurs
- It is possible to debug the Web application remotely
- It is possible to debug client scripts written in JavaScript

ASP.NET Tracing

- Tracing allows to view diagnostic information about a single request and to write debug statements directly in code
- There are two tracing options (it is possible to route messages between tracing mechanisms):
  - ASP.NET tracing (using the `trace.axd` address)
  - Standard .NET Framework trace output (the `System.Diagnostics.Trace` class)

Enabling Tracing for Pages

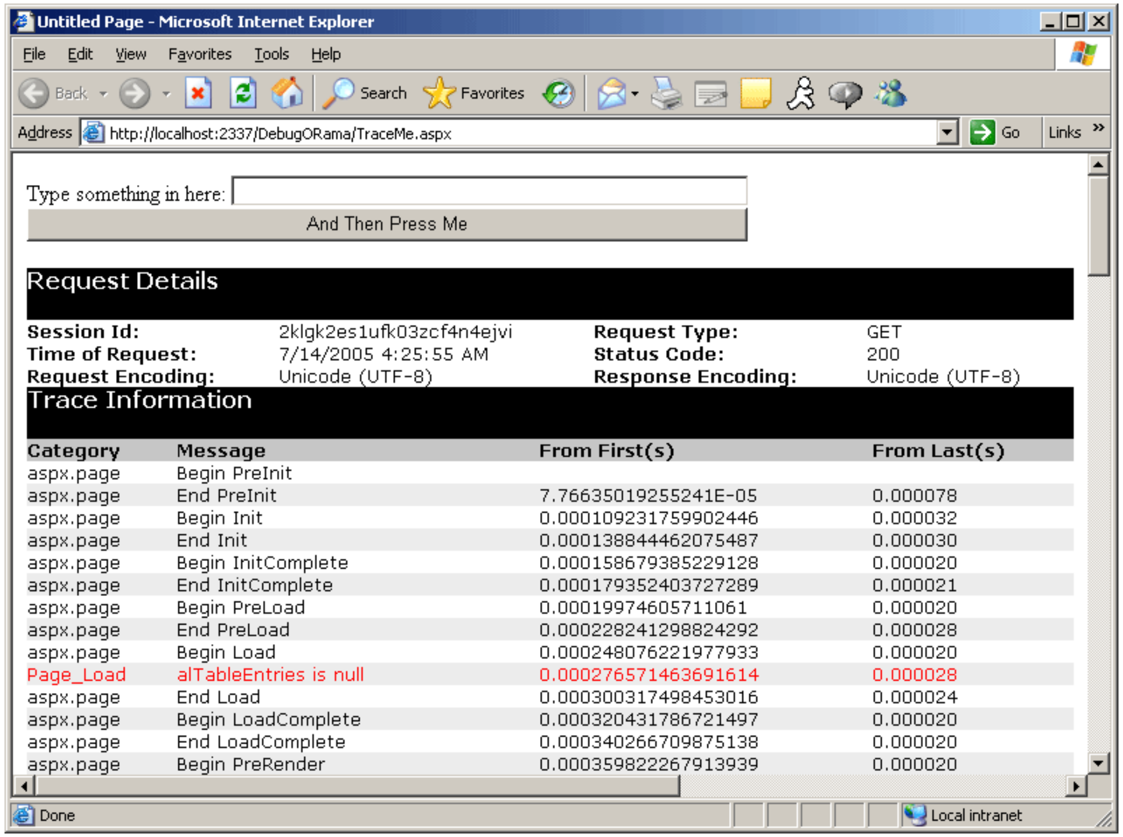
- When tracing is enabled for a page, trace information is displayed if any browser requests the page
- To enable tracing for a page set **`trace=true`** for the **`@Page`** directive
- Optionally, the **`TraceMode`** attribute can be used to specify the order in which trace messages appear
  - **`SortByTime, SortByCategory`**

```
<%@ Page Language="C#"
         Trace="True"
         TraceMode="SortByCategory" %>
```

- The **`Trace`** attribute in the **`@Page`** directive takes precedence over attributes set in the trace element in the **`Web.config`** file

Writing Trace Information

- Both the **`Trace.Write()`** and **`Trace.Warn()`** methods write trace information, the **`Warn()`** method uses red colour for text


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2337/DebugORama/TraceMe.aspx
Go Links
Done
Local intranet

Type something in here:

And Then Press Me

Request Details

Session Id:	2klgk2es1ufk03zcf4n4ejvi	**Request Type:**	GET
Time of Request:	7/14/2005 4:25:55 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	7.76635019255241E-05	0.000078
aspx.page	Begin Init	0.000109231759902446	0.000032
aspx.page	End Init	0.000138844462075487	0.000030
aspx.page	Begin InitComplete	0.000158679385229128	0.000020
aspx.page	End InitComplete	0.000179352403727289	0.000021
aspx.page	Begin PreLoad	0.00019974605711061	0.000020
aspx.page	End PreLoad	0.000228241298824292	0.000028
aspx.page	Begin Load	0.000248076221977933	0.000020
Page_Load	alTableEntries is null	0.000276571463691614	0.000028
aspx.page	End Load	0.000300317498453016	0.000024
aspx.page	Begin LoadComplete	0.000320431786721497	0.000020
aspx.page	End LoadComplete	0.000340266709875138	0.000020
aspx.page	Begin PreRender	0.000359822267913939	0.000020

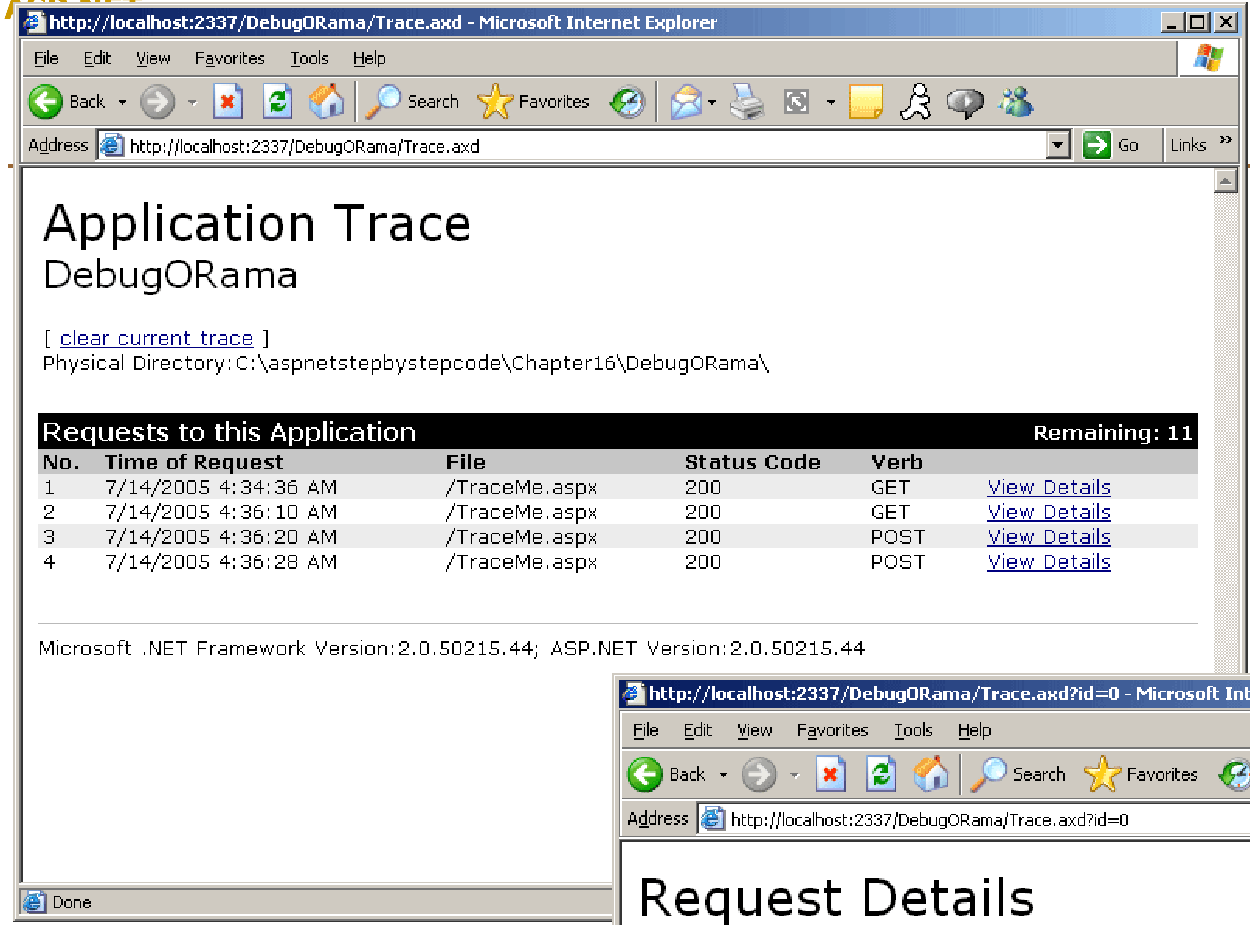
```
Trace.Warn("Page_Load",  "alTableEntries is null");
```

```
Trace.Write("Button1_Click", "Adding: " + TextBox1.Text);
```

Application-Level Tracing

```
<configuration>
  <system.web>
    <trace enabled="true" requestLimit="40" localOnly="false" />
  </system.web>
</configuration>
```

- The trace configuration attributes:
  - **`enabled`** – the default is **`false`**
  - **`pageOutput`** – the default is **`false`** (**`trace.axd`** can be used)
  - **`requestLimit`** – the number of trace requests to store on the server (the default is 10)
  - **`traceMode`** – **`SortByTime`**, **`SortByCategory`**
  - **`localOnly`** – **`true`** to make **`trace.axd`** available only on the server
  - **`mostRecent`** – **`true`** to display the most recent trace information as tracing output (the default is **`false`**)
  - **`writeToDiagnosticsTrace`** – **`true`** to use **`System.Diagnostics`**


http://localhost:2337/DebugORama/Trace.axd - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2337/DebugORama/Trace.axd
Go Links
Done

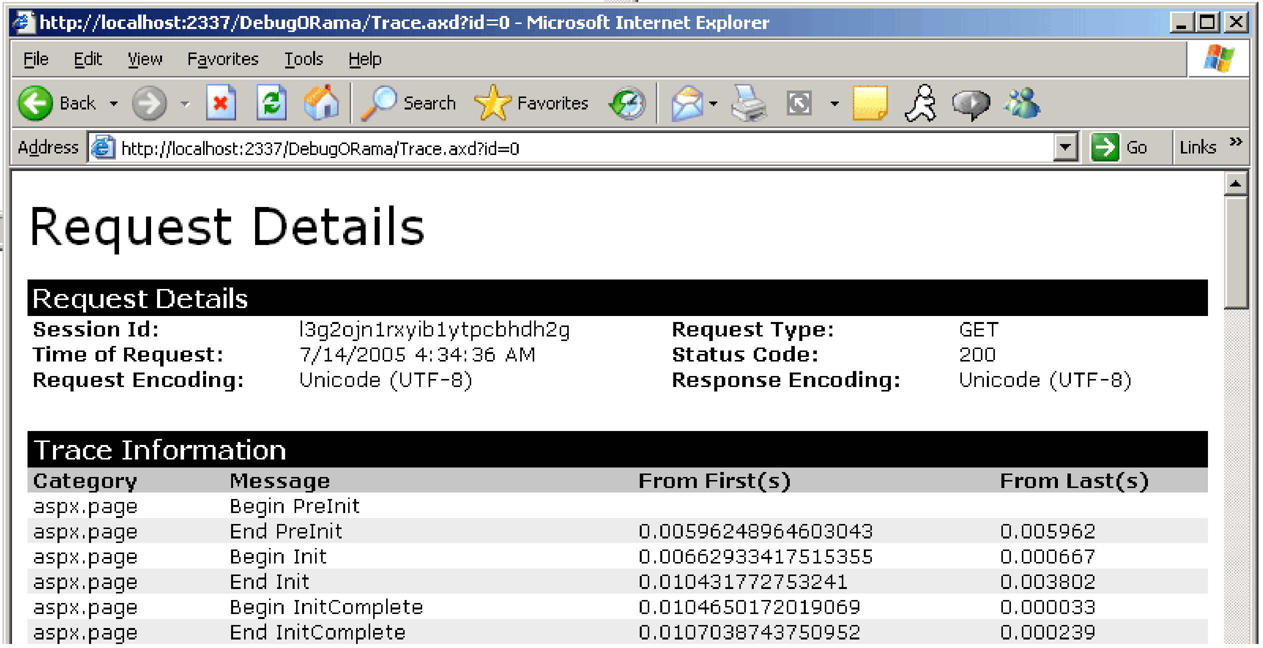
Application Trace
DebugORama

[clear current trace]
Physical Directory:C:\aspnetstepbystepcode\Chapter16\DebugORama\

Requests to this Application — Remaining: 11

No.	Time of Request	File	Status Code	Verb	
1	7/14/2005 4:34:36 AM	/TraceMe.aspx	200	GET	View Details
2	7/14/2005 4:36:10 AM	/TraceMe.aspx	200	GET	View Details
3	7/14/2005 4:36:20 AM	/TraceMe.aspx	200	POST	View Details
4	7/14/2005 4:36:28 AM	/TraceMe.aspx	200	POST	View Details

Microsoft .NET Framework Version:2.0.50215.44; ASP.NET Version:2.0.50215.44


http://localhost:2337/DebugORama/Trace.axd?id=0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2337/DebugORama/Trace.axd?id=0
Go Links

Request Details

Request Details

Session Id:	l3g2ojn1rxyib1ytpcbhdh2g	Request Type:	GET
Time of Request:	7/14/2005 4:34:36 AM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.00596248964603043	0.005962
aspx.page	Begin Init	0.00662933417515355	0.000667
aspx.page	End Init	0.010431772753241	0.003802
aspx.page	Begin InitComplete	0.0104650172019069	0.000033
aspx.page	End InitComplete	0.0107038743750952	0.000239

TraceListeners

```
TextWriterTraceListener myTextListener = new
TextWriterTraceListener(File.Create(@"C:\myListener.log"));
Trace.Listeners.Add(myTextListener);
```

- or -

```
<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="4">
      <listeners>
        <add name="myListener"
             type="System.Diagnostics.TextWriterTraceListener"
             initializeData="c:\myListener.log" />
        <remove name="Default" />
      </listeners>
    </trace>
  </system.diagnostics>
</configuration>
```

WebPageTraceListener

- New in ASP.NET 2.0
- Automatically forwards tracing information from any component calls to **`System.Diagnostics.Trace.Write`**
  - It allows to write components using the most generic trace provider and see its tracing output in the context of the application

```
<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="4">
      <listeners>
        <add name="webListener"
            type="System.Web.WebPageTraceListener, System.Web"/>
      </listeners>
    </trace>
  </system.diagnostics>
  <system.web>
    <trace enabled="true" pageOutput="false" localOnly="true" />
  </system.web>
</configuration>
```

Handling Page-Level Errors

- The **`Error`** event handler of the page catches all unhandled exceptions on the current page
  - It is preferable to use try/catch blocks
  - Error information cannot be displayed in a control (e.g. as a **`Label`** control) because new instances of controls are not created on the page when the **`Error`** handler is called
  - After handling an error, it must be cleared by calling the **`Server.ClearError()`** method

```
private void Page_Error(object sender, EventArgs e) {
  Response.Write("An application error has been logged.");
  ApplicationSpecificErrorLogger(Server.GetLastError().Message);
  Server.ClearError();
}
```

Handling Application-Level Errors

```
// Global.asax
void Application_Error(Object sender, EventArgs e) {
  Server.Transfer("Errors.aspx");
}
```

```
// Errors.aspx
protected void Page_Load(object sender, EventArgs e) {
  System.Text.StringBuilder errMessage = new StringBuilder();
  System.Exception appException = Server.GetLastError();
  if (appException is HttpException) {
    HttpException checkException = (HttpException)appException;
    switch (checkException.GetHttpCode()) {
      case 403: { errMessage.Append("Forbidden."); break; }
      case 404: { errMessage.Append("Not found"); break; }
      // ...
      default: { errMessage.Append(" An error."); break; }
    }
  } else {
    errMessage.AppendFormat("Error: <br />{0}",
                            appException.ToString());
  }
  Label1.Text = errMessage.ToString();
  Server.ClearError();
}
```

Custom Errors in Web.config

```
<configuration>
    <system.web>
        <customErrors defaultRedirect="CustomErrorPage.htm"
                      mode="On" >
            <error statusCode="400"
                   redirect="CustomErrorPage400.htm"/>
            <error statusCode="404"
                   redirect="CustomErrorPage404.htm"/>
            <error statusCode="500"
                   redirect="CustomErrorPage500.htm"/>
        </customErrors>
```

- **`defaultRedirect`** – default URL for errors
- **`mode`** – **`On`**, **`Off`**, **`RemoteOnly`** (custom errors are shown only to remote clients; the default)

Guidelines for Error Handling

1. Catch what you expect:
   - Use a try/catch blocks around error-prone code
   - Consider using the page-level error handler to catch specific exceptions that might happen anywhere on the page
2. But prepare for unhandled exceptions:
   - Set the `ErrorPage` property if a specific page should show a specific error at page for any unhandled exception
   - Have default error pages for 400 and 500 errors in the Web.config file
   - Have the `Application_OnError` handler that takes into consideration both specific exceptions, as well as all unhandled exceptions

ASP.NET Health Monitoring

- ASP.NET health monitoring allows system administrators to monitor the status of deployed Web applications
- There are a number of built-in events, including application lifetime events such as start and stop and a heartbeat event
- General scenarios:
  - Monitoring the performance of an application to ensure that it is healthy
  - Rapidly diagnosing failing applications or systems
  - Appraising significant events during the lifecycle of a given application

Web Events

- Web events can contain information about the ASP.NET worker process, application domain, request data, response data, application errors, and configuration errors
- By default, ASP.NET automatically records some Web event data
- Web events can be customized:
  - The number of event notifications that occur in a given time span can be limited
  - An existing provider or custom provider can be subscribed to a Web event
  - Custom events can be created by inheriting from existing event classes

Caching

ASP.NET Caching

- ASP.NET provides caching using two basic caching mechanisms:
  - The page output caching, which saves the output of page processing and reuses the output when a user requests the page again
  - The application caching, which allows to cache any generated data (e.g. `DataSets`)

Caching ASP.NET Pages

- Cache settings can be specified declaratively in a page or configuration file, or programmatically using a cache API
- It is possible to cache pages based on the values of query string parameters or form variables
- ASP.NET allows to write code during testing if cached content should be served
- It is possible to cache a portion of a page

Setting the Cacheability

```
<%@ OutputCache Duration="60" VaryByParam="None"%>
```

- or -

```
<caching>
  <outputCacheSettings>
    <outputCacheProfiles>
      <add name="Cache30Seconds" duration="30"
           varyByParam="none" />
    </outputCacheProfiles>
  </outputCacheSettings>
</caching>
```

```
<%@ OutputCache CacheProfile="Cache30Seconds" %>
```

- or -

```
Response.Cache.SetCacheability(HttpCacheability.Public)
```

Checking the Validity

```
public static void ValidateCacheOutput(HttpContext context,
                Object data, ref HttpValidationStatus status) {
  if (context.Request.QueryString["Status"] != null) {
    string pageStatus = context.Request.QueryString["Status"];
    if (pageStatus == "invalid") {
      status = HttpValidationStatus.Invalid;
    } else if (pageStatus == "ignore") {
      status = HttpValidationStatus.IgnoreThisRequest;
    } else {
      status = HttpValidationStatus.Valid;
  } else {
    status = HttpValidationStatus.Valid;
  }
}

protected void Page_Load(object sender, EventArgs e) {
  Response.Cache.AddValidationCallback(
        new HttpCacheValidateHandler(ValidateCacheOutput),
        null);
}
```

Using Dependencies

- Dependency on a file or files

```
protected void Page_Load(object sender, EventArgs e) {
    string fileDependencyPath = Server.MapPath("TextFile1.txt");
    Response.AddFileDependency(fileDependencyPath);
    // Set additional properties to enable caching.
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
    Response.Cache.SetCacheability(HttpCacheability.Public);
    Response.Cache.SetValidUntilExpires(true);
}
```

- Dependency on other item in the cache

```
protected void Page_Load(object sender, EventArgs e) {
    Response.AddCacheItemDependency("ProcessIntensiveReport");
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(60));
    Response.Cache.SetCacheability(HttpCacheability.Public);
    Response.Cache.SetValidUntilExpires(true);
}
```

Caching Application Data

- The **`Cache`** class implements a powerful, easy-to-use caching mechanism that allows to store objects in memory that require extensive server resources to create
- Available features:
  - Expiration policies (both absolute and sliding expiration time can be used)
  - A file or key dependency
    - The **`SqlCacheDependency`** class can be used to create a cache item dependency on a table or row in a database
    - The **`CacheDependency`** class is unsealed (it allows to create custom dependencies)
  - Priorities

Adding Items to Application Cache

```
Cache["CacheItem1"] = "Cached Item 1";
```

```
string[] dependencies = { "CacheItem2" };
Cache.Insert("CacheItem3", "Cached Item 3",
    new System.Web.Caching.CacheDependency(null, dependencies));
```

```
Cache.Insert("CacheItem4", "Cached Item 4",
    new System.Web.Caching.CacheDependency(
    Server.MapPath("XMLFile.xml")));
```

```
Cache.Insert("CacheItem7", "Cached Item 7",
    null, System.Web.Caching.Cache.NoAbsoluteExpiration,
    new TimeSpan(0, 10, 0));
```

```
Cache.Insert("CacheItem8", "Cached Item 8",
    null, System.Web.Caching.Cache.NoAbsoluteExpiration,
    System.Web.Caching.Cache.NoSlidingExpiration,
    System.Web.Caching.CacheItemPriority.High, null);
```

Reading and Deleting Items

- Reading an item from the cache:

```
string cachedString;
if (Cache["CacheItem"] != null) {
    cachedString = (string)Cache["CacheItem"];
} else {
    Cache.Insert("CacheItem", "Hello, World.");
    cachedString = (string)Cache["CacheItem"];
}
```

- Deleting an item from the cache:

```
Cache.Remove("MyData1");
```

Items Removing Notification

```
HttpContext.Current.Cache.Add("MyReport", CreateReport(), null,
    DateTime.MaxValue, new TimeSpan(0, 1, 0),
    System.Web.Caching.CacheItemPriority.Default,
    ReportRemovedCallback);
```

```
public static void ReportRemovedCallback(String key,
          object value, CacheItemRemovedReason removedReason) {
    _reportRemovedFromCache = true;
    CacheReport();
}
```

Caching Multiple Versions of a Page

- ASP.NET allows to cache multiple versions of the same page in the output cache
- The `@OutputCache` directive includes four attributes that enable to cache multiple versions of the page output
  - `VaryByParam` – dependency on the query string
  - `VaryByControl` – dependency on a control value
  - `VaryByHeader` – dependency on the request's HTTP header
  - `VaryByCustom` – by a browser type or by a custom string

```
<%@ OutputCache Duration="60" VaryByParam="City" %>
```

```
Response.Cache.SetExpires(DateTime.Now.AddMinutes(1.0));
Response.Cache.SetCacheability(HttpCacheability.Public);
Response.Cache.SetValidUntilExpires(true);
Response.Cache.VaryByParams["Zip"] = true;
```

Caching Portions of a Page

- The control caching (aka fragment caching) allows to cache parts of the page output
  - Caching policies for the control can be created declaratively using the **`@OutputCache`** directive or by using the **`PartialCachingAttribute`** in the code

```
<%@ OutputCache Duration="120" VaryByParam="None" %>
```

```
[PartialCaching(120)]
public partial class CachedControl : System.Web.UI.UserControl {
    // ...
}
```

Post-Cache Substitution

- ASP.NET postback substitution allows to cache a page but substitute some content dynamically
  - The entire page is output cached with specific parts marked as exempt from caching
- There are three ways to implement post-cache substitution:
  - Declaratively, using the **`Substitution`** control
  - Programmatically, using the Substitution control API
  - Implicitly, using the **`AdRotator`** control
    - The **`AdRotator`** control placed on a cached page renders unique advertisements on each request
    - It implements support for post-cache substitution internally

ASP.NET Performance Issues

- Session state – if the application does not need it, disable it
- View state – by default enabled for all controls, it consumes bandwidth and takes time to process
- Caching – use output and data caching whenever possible
- Server controls – if you do not need to manipulate a control programmatically, do not use a server control (use a classic HTML control instead)
- Web gardening and Web farming – at the least, locate the Web server on one machine and the database server on another
- Round trips – most validation and data manipulations can occur on the client browser

Security

Basic Security Practices

- General Web application security recommendations
  - Back up often and keep your backups physically secure
  - Keep your Web server computer physically secure
  - Use the Windows NTFS file system, not FAT32
  - Secure the Web server computer and all computers on the same network with strong passwords
  - Secure IIS
  - Close unused ports and turn off unused services
  - Run a virus checker that monitors inbound and outbound traffic
  - Use a firewall
  - Install the latest security patches from Microsoft and other vendors
  - Use Windows event logging and examine the logs frequently for suspicious activity

Basic Security Practices cont.

- Run applications with least privileges
  - Do not run your application with the identity of a system user (administrator)
  - Run the application in the context of a user with the minimum practical privileges
  - Set permissions on all the resources required for your application
  - Keep files for your Web application in a folder below the application root
- Know your users
  - If your application is an intranet application, configure it to use Windows integrated security
  - If you need to gather credentials from the user, use one of the ASP.NET authentication strategies

Basic Security Practices cont.

- Guard against malicious user input
  - Filter user input to check for HTML tags, which might contain script
  - Never echo (display) unfiltered user input, always use HTML encoding
  - Never store unfiltered user input in a database
  - If you want to accept some HTML from a user, filter it manually
  - Do not assume that information you get from the HTTP request header is safe
    - Use safeguards for query strings, cookies, and so on
  - If possible, do not store sensitive information in a place accessible from the browser, such as hidden fields or cookies

Basic Security Practices cont.

- Access database securely
  - Use the inherent security of your database to limit who can access database resources
    - If practical in your application, use integrated security
    - If your application involves anonymous access, create a single user with very limited permissions, and perform queries by connecting as this user
  - Do not create SQL statements by concatenating strings that involve user input
    - Create a parameterized query and use user input to set parameter values
  - If you must store a user name and password somewhere to use as the database login credentials, store them in the Web.config file and secure the file with protected configuration

Basic Security Practices cont.

- Create safe error messages
  - Do not write error messages that echo information that might be useful to malicious users, such as a user name
  - Configure the application not to show detailed errors to users
  - Use the **`customErrors`** configuration element to control who can view exceptions from the server
  - Create custom error handling for situations that are prone to error, such as database access

Basic Security Practices cont.

- Keep sensitive information safely
  - If your application transmits sensitive information between the browser and the server, consider using the Secure Sockets Layer (SSL)
  - Use protected configuration to secure sensitive information in configuration files such as the **`Web.config`** or **`Machine.config`** files
  - If you must store sensitive information, do not keep it in a Web page, even in a form that you think people will not be able to see it (such as in server code)
  - Use the strong encryption algorithms supplied in the **`System.Security.Cryptography`** namespace

Basic Security Practices cont.

- Use cookies securely
  - Do not store any critical information in cookies
  - Set expiration dates on cookies to the shortest practical time you can; avoid permanent cookies if possible
  - Consider encrypting information in cookies
  - Consider setting the **`Secure`** and **`HttpOnly`** properties on the cookie to **`true`** (to use SSL and don't allow to access the cookie through a client-side script)

Basic Security Practices cont.

- Guard against denial-of-service threats
  - Use error handling, include a finally block in which you release resources in case of failure
  - Configure IIS to use process throttling
  - Test size limits of user input before using or storing it
  - Put size safeguards on database queries
  - Put a size limit on file uploads

Configuration Management

ASP.NET Configuration System

- Using the features of the ASP.NET configuration system, the following elements can be configured:
  - An entire server
  - An ASP.NET application
  - Individual pages
  - Application subdirectories
- Configuration files
  - The root of the ASP.NET configuration hierarchy is: ***`systemroot`***`\Microsoft.NET\Framework\`***`version`***`\CONFIG\Web.config`
  - **`Web.config`** files can appear in multiple directories in ASP.NET applications

Configuration Inheritance

- All configuration files that are located in the virtual directory path for the requested URL are used to compute the configuration settings for the request
  - The most local configuration settings override settings in parent configuration files
  - The configuration settings can be locked by adding the **`allowOverride`** attribute to the **`location`** element

```
<configuration>
    <location path="application1" allowOverride="false">
        <system.web>
            <trust level="High" />
        </system.web>
    </location>
</configuration>
```

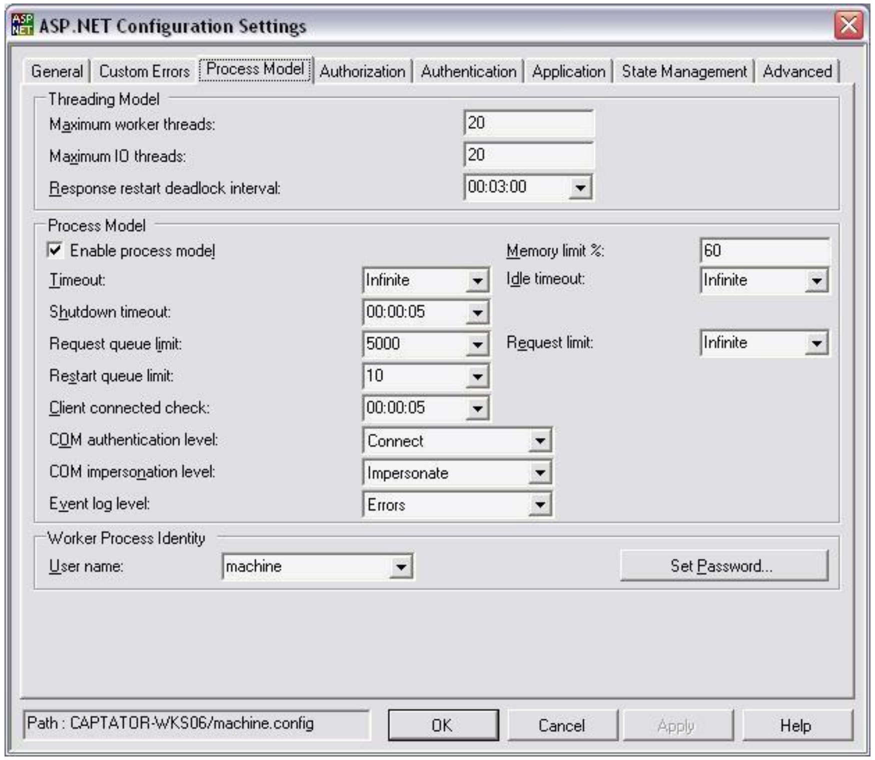
  - The location element allows to apply configuration settings to specific folders and files

Configuration Tools

- ASP.NET MMC (Microsoft Management Console) snap-in
  - Available as the properties window in the IIS configuration
  - It provides a convenient way to manipulate ASP.NET configuration settings at all levels on a local or remote Web server
- Command-line tools
  - aspnet_compiler.exe – allows to compile the application
  - aspnet_regiis.exe – allows to easily update the script maps for an ASP.NET application

ASP.NET MMC Snap-In

- Information:
  - The ASP.NET version, virtual path, file location, file creation date, file modification date
- Settings:
  - Connection strings, app settings, custom errors, threading model, process model, authorization, authentication, compilation, globalization, identity, state management, locations


ASP.NET Configuration Settings

General | Custom Errors | Process Model | Authorization | Authentication | Application | State Management | Advanced

Threading Model

Maximum worker threads:	20
Maximum IO threads:	20
Response restart deadlock interval:	00:03:00

Process Model

☑ Enable process model		Memory limit %:	60
Timeout:	Infinite	Idle timeout:	Infinite
Shutdown timeout:	00:00:05		
Request queue limit:	5000	Request limit:	Infinite
Restart queue limit:	10		
Client connected check:	00:00:05		
COM authentication level:	Connect		
COM impersonation level:	Impersonate		
Event log level:	Errors		

Worker Process Identity

User name: machine

Path : CAPTATOR-WKS06/machine.config

ASP.NET Configuration API

- The **`Configuration`** class represents configurations of computers, .NET client applications, Web directories, and resources that are stored in Web directories
  - In ASP.NET 2.0 the methods of the **`WebConfigurationManager`** should be used to gain access to an instance of the **`Configuration`** class
  - The specified logical entity can exist on the local computer or on a remote server

WebConfigurationManager Key Members

- `AppSettings`, `ConnectionStrings`, `GetSection()`, `GetWebApplicationSection()` – gets data for the current Web application's default configuration
- `OpenMachineConfiguration()`, `OpenMappedMachineConfiguration()` – gets the `Configuration` object representing the machine-configuration file
- `OpenWebConfiguration()`, `OpenMappedWebConfiguration()` – gets the `Configuration` object representing the Web-application configuration file

Using WebConfigurationManager

```
System.Configuration.Configuration config =
    WebConfigurationManager.OpenWebConfiguration("/configTest")
    as System.Configuration.Configuration;
```

```
System.Configuration.Configuration config =
    WebConfigurationManager.OpenMachineConfiguration();
```

```
AppSettingsSection appSettingsSection =
    WebConfigurationManager.GetSection("appSettings")
    as AppSettingsSection;
```

```
string s = (string)WebConfigurationManager.AppSettings["theKey"];
```

```
ConnectionStringsSection connectionStringsSection =
    WebConfigurationManager.GetSection("connectionStrings",
    "/configTest") as ConnectionStringsSection;
```

```
Configuration config =
    WebConfigurationManager.OpenWebConfiguration("/MyApp");
config.SaveAs("c:\\MyApp.web.config",
    ConfigurationSaveMode.Full, true);
```

Packaging and Deploying

Deployment Pieces

- Parts of the Web application which need deployment consideration when moving the application:
  - .aspx pages
  - The code-behind pages for the .aspx pages (.aspx.vb or .aspx.cs files)
  - User controls (.ascx)
  - Web service files (.asmx and .wsdl files)
  - .htm or .html files
  - Image files such as .jpg or .gif
  - ASP.NET system folders such as App_Code and App_Themes
  - JavaScript files (.js)
  - Cascading Style Sheets (.css)
  - Configuration files such as the web.config file
  - .NET components and compiled assemblies
  - Data files such as .mdb files

Steps Before Deploying

1. Turn off debugging in the Web.config file

```
<configuration xmlns="http://schemas.microsoft.com/
                      .NetConfiguration/v2.0">
    <system.web>
        <compilation debug="false" />
    </system.web>
</configuration>
```

2. Build the application in Release mode

Methods of Deploying Web Applications

- XCopy
- Visual Studio 'Publish Web Site' option
  - Optional precompilation of all application's files
- Using Windows Installer
  - Installation, removal, and management of applications
  - Automatic repair of existing installations
  - Transactional operations
  - Installation on demand
  - Installation in locked-down environment
  - 'Web Setup Project' is available in Visual Studio