UNIT-IV

SYLLABUS

Goal of ASP.NET –ASP.NET Web Server Control-Validation Server Controls-Themes and Skins -Content Page Holder

ASP.NET

- ASP-Active server page
- Active Server Page program is a Server-Side Script (a program running on a Server) to create the widely used Microsoft operating system limit on Microsoft Windows by using the Web.
- Writing ASP.NET provides a family file ending with .aspx ASP.NET which language can be used to control database programming and many other languages are ASP.NET Microsoft has focused to develop a language.
- Generation of the world's next generation Internet-called new generation or that generation Internet Web2.0 a new generation to replace the older generation Internet (Web1.0).
- ASP is a language created by starting from the Active Server Page 3.0 (ASP 3.0) and develop a ASP.NET 1.0 / 1.1 / 2.0 / to ASP.NET 3.5, respectively, which is an optimization web development of applications.

PURPOSE OF ASP.NET:

- The main purpose of ASP.NET 3.5 is to allow users to create web applications that are secure, convenient and easy to reduce the number of write less code because version control is added.
- Accessing more functions.

Capabilities of ASP.NET:

ASP.NET can take the technology. NET Framework to be able to use apple pin with applications such as computer hardware any Palm PDA Notebook and mobile phones etc..

Make a web page developed. Technology in the form of ASP.NET Web Form is divided into 2 sections of the tag is used and the display processing program used. The screen is similar to that used tools. In

KAHE

developing programs such as Visual Basic and C + +, etc.

Web Browser with all the different types of orders due. Defined in the Web Form, it is converted to HTML tag with an appropriate Web Browser, which differs from ASP in some of the traditional command does not work in certain Web Browser.

Support for working with the program. Developed from the language . NET such as VB.NET and C #. NET is.

The goal of ASP.NET 3.5

- Microsoft has targeted the development team of ASP.NET 3.5 with focus on what is useful to developers, administrators and management with the added performance.
- Code to eliminate the surplus in the previous version of ASP.NET. Make it easy to retrieve data such as shown in the table format control for ASP.NET 3.5 is called "**GridView Server Control**" .
- The GridView Server Control is capable of formatting the page [Paging] Sort of. order [Sorting] to correct this. To reduce the steps of writing code.
- One key goal of ASP.NET 3.5 is to speed up Web applications apple pin. One of ability is. The ability to cache [Cache] data with Microsoft SQL Server in ASP.NET 3.5 that includes critical characteristics called "**SQL Cache Invalidation"** helping to provide the information that a User would like to save the current time.
- This ASP.NET 3.5 to add support 64 - bit, which means users can apple Rapid Run ASP.NET applications on your Intel or AMD Processor Size 64 - bit and also compatible with ASP.NET 1.0,. 1.1 and 2.0 can be variations in apple Rapid applications of ASP.NET 1.0,1.1, and 2.0 to the. NET Framework 3.5.

Web Server Controls:

- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.
- The syntax for creating a Web server control is:

```
<ASP:CONTROL_NAME  ID="SOME_ID"  RUNAT="SERVER"  />
```

KAHE

Web Server Control	Description
AdRotator	Displays a sequence of images
Button	Displays a push button
Calendar	Displays a calendar
CalendarDay	A day in a calendar control
CheckBox	Displays a check box
CheckBoxList	Creates a multi-selection check box group
DataGrid	Displays fields of a data source in a grid
DataList	Displays items from a data source by using templates
DropDownList	Creates a drop-down list
HyperLink	Creates a hyperlink
Image	Displays an image
ImageButton	Displays a clickable image
Label	Displays static content which is programmable (lets you apply styles to its content)
LinkButton	Creates a hyperlink button
ListBox	Creates a single- or multi-selection drop-down list
ListItem	Creates an item in a list
Literal	Displays static content which is programmable(does not let you apply styles to its content)
Panel	Provides a container for other controls
PlaceHolder	Reserves space for controls added by code

KAHE

RadioButton	Creates a radio button
RadioButtonList	Creates a group of radio buttons
BulletedList	Creates a list in bullet format
Repeater	Displays a repeated list of items bound to the control
Style	Sets the style of controls
Table	Creates a table
TableCell	Creates a table cell
TableRow	Creates a table row
TextBox	Creates a text box
Xml	Displays an XML file or the results of an XSL transform

AdRotator Control

The AdRotator control is used to display a sequence of ad images.

This control uses an XML file to store the ad information. The XML file must begin and end with an <Advertisements> tag. Inside the <Advertisements> tag there may be several <Ad> tags which defines each ad.

The predefined elements inside the <Ad> tag are listed below:

Element	Description
<ImageUrl>	Optional. The path to the image file
<NavigateUrl>	Optional. The URL to link to if the user clicks the ad
<AlternateText>	Optional. An alternate text for the image
<Keyword>	Optional. A category for the ad
<Impressions>	Optional. The display rates in percent of the hits

The Button Control

The Button control is used to display a push button. The push button may be a submit button or a command button. By default, this control is a submit button.

A submit button does not have a command name and it posts the Web page back to the server when it is clicked. It is possible to write an event handler to control the actions performed when the submit button is clicked.

A command button has a command name and allows you to create multiple Button controls on a page. It is possible to write an event handler to control the actions performed when the command button is clicked.

Properties

Property	Description
CausesValidation	Specifies if a page is validated when a button is clicked
CommandArgument	Specifies additional information about the command to perform
CommandName	Specifies the command associated with the Command event
OnClientClick	Specifies the name of the function to be executed when a button is clicked

PostBackUrl	Specifies the URL of the page to post to from the current page when a button is clicked
runat	Specifies that the control is a server control. Must be set to "server"
Text	Specifies the text on a button
UseSubmitBehavior	Specifies whether or not a button uses the browser's submit mechanism or the ASP.NET postback mechanism
ValidationGroup	Specifies the group of controls a button causes validation, when it posts back to the server

The Calendar control

The Calendar control is used to display a calendar in the browser.

This control displays a one-month calendar that allows the user to select dates and move to the next and previous months.

SelectedDate	The selected date
SelectedDates	The selected dates
SelectedDayStyle	The style for selected days
SelectionMode	How a user is allowed to select dates
SelectMonthText	The text displayed for the month selection link
SelectorStyle	The style for the month and weeks selection links
SelectWeekText	The text displayed for the week selection link

CheckBox Control

The CheckBox control is used to display a check box.

Properties

Property	Description
AutoPostBack	Specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
CausesValidation	Specifies if a page is validated when a Button control is clicked
Checked	Specifies whether the check box is checked or not
InputAttributes	Attribute names and values used for the Input element for the CheckBox control
LabelAttributes	Attribute names and values used for the Label element for the CheckBox control
runat	Specifies that the control is a server control. Must be set to "server"
Text	The text next to the check box
TextAlign	On which side of the check box the text should appear (right or left)
ValidationGroup	Group of controls for which the Checkbox control causes validation when it posts back to the server
OnCheckedChanged	The name of the function to be executed when the Checked property has changed

CheckBoxList Control

The CheckBoxList control is used to create a multi-selection check box group.

Each selectable item in a CheckBoxList control is defined by a ListItem element!

Properties

Property	Description
CellPadding	The amount of pixels between the border and the contents of the table cell
CellSpacing	The amount of pixels between table cells
RepeatColumns	The number of columns to use when displaying the check box group
RepeatDirection	Specifies whether the check box group should be repeated horizontally or vertically
RepeatLayout	The layout of the check box group
runat	Specifies that the control is a server control. Must be set to "server"
TextAlign	**On which side of the check box the text should appear**

DropDownList Control

The DropDownList control is used to create a drop-down list.

Each selectable item in a DropDownList control is defined by a ListItem element!

Properties

Property	Description
SelectedIndex	The index of a selected item
OnSelectedIndexChanged	The name of the function to be executed when the index of the selected item has changed
runat	Specifies that the control is a server control. Must be set to "server"

ListBox Control

The ListBox control is used to create a single- or multi-selection drop-down list.

Each selectable item in a ListBox control is defined by a ListItem element!

Properties

Property	Description
Rows	The number of rows displayed in the list
SelectionMode	Allows single or multiple selections

RadioButton Control

The RadioButton control is used to display a radio button.

Properties

Property	Description
AutoPostBack	A Boolean value that specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
Checked	A Boolean value that specifies whether the radio button is checked or not
id	A unique id for the control
GroupName	The name of the group to which this radio button belongs
OnCheckedChanged	The name of the function to be executed when the Checked property has changed
runat	Specifies that the control is a server control. Must be set to "server"
Text	The text next to the radio button
TextAlign	On which side of the radio button the text should appear (right or left)

RadioButtonList Control

The RadioButtonList control is used to create a group of radio buttons.

Each selectable item in a RadioButtonList control is defined by a ListItem element.

Property	Description
CellPadding	The amount of pixels between the border and the contents of the table cell
CellSpacing	The amount of pixels between table cells

RepeatColumns	The number of columns to use when displaying the radio button group
RepeatDirection	Specifies whether the radio button group should be repeated horizontally or vertically
RepeatLayout	The layout of the radio button group
runat	Specifies that the control is a server control. Must be set to "server"
TextAlign	On which side of the radio button the text should appear

Table Control

The Table control is used in conjunction with the TableCell control and the TableRow control to create a table.

Property	Description
BackImageUrl	A URL to an image to use as an background for the table
Caption	The caption of the table
CaptionAlign	The alignment of the caption text
CellPadding	The space between the cell walls and contents
CellSpacing	The space between cells
GridLines	The gridline format in the table
HorizontalAlign	The horizontal alignment of the table in the page

runat	Specifies that the control is a server control. Must be set to "server"

TextBox Control

The TextBox control is used to create a text box where the user can input text.

Property	Description
AutoCompleteType	Specifies the AutoComplete behavior of a TextBox
AutoPostBack	A Boolean value that specifies whether the control is automatically posted back to the serve when the contents change or not. Default is false
CausesValidation	Specifies if a page is validated when a Postback occurs
Columns	The width of the textbox
MaxLength	The maximum number of characters allowed in the textbox
ReadOnly	Specifies whether or not the text in the text box can be changed
Rows	The height of the textbox (only used if TextMode="Multiline")
runat	Specifies that the control is a server control. Must be set to "server"
TagKey	
Text	The contents of the textbox

TextMode	Specifies the behavior mode of a TextBox control (SingleLine, MultiLine or Password)
ValidationGroup	The group of controls that is validated when a Postback occurs
Wrap	A Boolean value that indicates whether the contents of the textbox should wrap or not
OnTextChanged	The name of the function to be executed when the text in the textbox has changed

Validation Server Controls:

- A Validation server control is used to validate the data of an input control.
- If the data does not pass validation, it will display an error message to the user.
- The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat="server" />
```

Validation Server Control	**Description**
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
CustomValidator	Allows you to write a method to handle the validation of the value entered
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
RequiredFieldValidator	Makes an input control a required field
ValidationSummary	Displays a report of all validation errors occurred in a Web page

KAHE

The RequiredFieldValidator:

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax for the control:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
       runat="server" ControlToValidate ="ddlcandidate"
       ErrorMessage="Please choose a candidate"
       InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

The RangeValidator:

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	**Description**
Type	it defines the type of the data; the available values are: Currency, Date, Double, Integer and String
MinimumValue	it specifies the minimum value of the range
MaximumValue	it specifies the maximum value of the range

The syntax for the control:

```
<asp:RangeValidator ID="rvclass"
      runat="server"
      ControlToValidate="txtclass"
      ErrorMessage="Enter your class (6 - 12)"
      MaximumValue="12"
      MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

The CompareValidator:

The CompareValidator control compares a value in one control with a fixed value, or, a value in another control.

It has the following specific properties:

Properties	Description
Type	it specifies the data type
ControlToCompare	it specifies the value of the input control to compare with
ValueToCompare	it specifies the constant value to compare with
Operator	it specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual and DataTypeCheck

The basic syntax for the control:

```
<asp:CompareValidator ID="CompareValidator1"
      runat="server"
      ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

The RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern against a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace
\t	Matches a tab
\r	Matches a carriage return
\v	Matches a vertical tab
\f	Matches a form feed
\n	Matches a new line
\	Escape character

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n
[abcd]	Matches any character in the set
[^abcd]	Excludes any character in the set
[2-7a-mA-M]	Matches any character specified in the range
\w	Matches any alphanumeric character and underscore
\W	Matches any non-word character

\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character
\d	Matches any decimal character
\D	Matches any non-decimal character

Quantifiers could be added to specify number of times a character could appear

Quantifier	Description
*	Zero or more matches
+	One or more matches
?	Zero or one matches
{N}	N matches
{N,}	N or more matches
{N,M}	Between N and M matches

The syntax for the control:

```
<asp:RegularExpressionValidator ID="string"
      runat="server"
      ErrorMessage="string"
      ValidationExpression="string"
      ValidationGroup="string">
</asp:RegularExpressionValidator>
```

The CustomValidator:

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, like JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control.s ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control

```
<asp:CustomValidator ID="CustomValidator1"
   runat="server"
   ClientValidationFunction=.cvf_func.
   ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

The ValidationSummary Control

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary:** shows the error messages in specified format.
- **ShowMessageBox:** shows the error messages in a separate window.

The syntax for the control:

<asp:ValidationSummary ID="ValidationSummary1"

```
   runat="server"
   DisplayMode = "BulletList"
   ShowSummary = "true"
   HeaderText="Errors:" />
```

Validation Groups:

Complex pages have different groups of information provided in different panels. In such a situation a need for performing validation separately for separate group, might arise. This kind of situation is handled using validation groups.

Themes and Skins:

- A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server.
- Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources.
- At a minimum, a theme will contain skins. Themes are defined in special directories in your Web site or on your Web server.

To apply a theme to a Web site

1. In the application's Web.config file, set the <pages> element to the name of the theme, either a global theme or a page theme, as shown in the following example:
2. <configuration>
3.   <system.web>
4.   <pages theme="ThemeName" />
5.   </system.web>

</configuration>

Skins:

- A skin file has the file name extension .skin and contains property settings for individual controls such as 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 part of the theme.
- For example, the following is a control skin for a Button control:

  Copy

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

KAHE

- You create .skin files in the Theme folder. A .skin file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file.
- There are two types of control skins:

  1)default skins

  2)named skins

Default skins:

A default skin automatically applies to all controls of the same type when a theme is applied to a page.

- A control skin is a default skin if it does not have a SkinID attribute.
- For example, if you create a default skin for a Calendar control, the control skin applies to all Calendar controls on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all Button controls, but not to LinkButton controls or to controls that derive from the Button object.)

Named skins:

- A named skin is a control skin with a SkinID property set.
- Named skins do not automatically apply to controls by type.
- Instead, you explicitly apply a named skin to a control by setting the control's SkinID property.
- Creating named skins allows you to set different skins for different instances of the same control in an application.

Applying Skins to Controls

Skins defined in your theme apply to all control instances in the application or pages to which the theme is applied. In some cases, you might want to apply a specific set of properties to an individual control. You can do that by creating a named skin (an entry in a .skin file that has a **SkinID** property set) and then applying it by ID to individual controls.

To apply a named skin to a control

KAHE

- Set the control's SkinID property, as shown in the following example:
  <asp:Calendar runat="server" ID="DatePicker" SkinID="SmallCalendar" />

To disable themes for a page

- Set the **EnableTheming** attribute of the @ Page directive to **false**, as in this example:
- <%@ Page EnableTheming="false" %>

To disable themes for a control

- Set the **EnableTheming** property of the control to **false**, as in this example:
  <asp:Calendar id="Calendar1" runat="server" EnableTheming="false" />

To apply a style sheet theme programmatically

- In the page's code, override the StyleSheetTheme property and in the **get** accessor, return the name of the style sheet theme.
  The following code example shows how to set a theme named BlueTheme as the style sheet theme for a page:

```
Public Overrides Property StyleSheetTheme() As String
  Get
     Return "BlueTheme"
  End Get
  Set(ByVal value As String)
  End Set
End Property
```

To apply control skins programmatically

- In a handler for the page's PreInit method, set the control's **SkinID** property.
  The following code example shows how to set the SkinID property of a Calendar control. This example assumes that the the page's theme has already been set.

KAHE

VB

```
Sub Page_PreInit(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
      Handles Me.PreInit
   Calendar1.SkinID = "CustomSkin"
End Sub
```

Cascading Style Sheets

- A theme can also include a cascading style sheet (.css file).
- When you put a .css file in the theme folder, the style sheet is applied automatically as part of the theme.
- You define a style sheet using the file name extension .css in the theme folder.

Theme Graphics and Other Resources

- Themes can also include graphics and other resources, such as script files or sound files. For example, part of your page theme might include a skin for a TreeView control.
- As part of the theme, you can include the graphics used to represent the expand button and the collapse button.
- Typically, the resource files for the theme are in the same folder as the skin files for that theme, but they can be elsewhere in the Web application, in a subfolder of the theme folder for example. To refer to a resource file in a subfolder of the theme folder, use a path like the one shown in this Image control skin:

```
<asp:Image runat="server" ImageUrl="ThemeSubfolder/filename.ext" />
```

- You can also store your resource files outside the theme folder.
- If you use the tilde (~) syntax to refer to the resource files, the Web application will automatically find the images. For example, if you place the resources for a theme in a subfolder of your application, you can use paths of the form ~/SubFolder/filename.ext to refer to resource files, as in the following example.

```
<asp:Image runat="server" ImageUrl="~/AppSubfolder/filename.ext" />
```

Scoping Themes

KAHE

- You can define themes for a single Web application, or as global themes that can be used by all applications on a Web server.
- After a theme is defined, it can be placed on individual pages using the Theme or StyleSheetTheme attribute of the @ Page directive, or it can be applied to all pages in an application by setting the <pages> element in the application configuration file.
- If the <pages> element is defined in the Machine.config file, the theme will apply to all pages in Web applications on the server.

Page Themes

- A page theme is a theme folder with control skins, style sheets, graphics files and other resources created as a subfolder of the \App_Themes folder in your Web site.
- Each theme is a different subfolder of the \App_Themes folder.
- The following example shows a typical page theme, defining two themes named BlueTheme and PinkTheme.

Global Themes

- A global theme is a theme that you can apply to all the Web sites on a server.
- Global themes allow you to define an overall look for your domain when you maintain multiple Web sites on the same server.
- Global themes are like page themes in that they include property settings, style sheet settings, and graphics.
- However, global themes are stored in a folder named Themes that is global to the Web server.
- Any Web site on the server, and any page in any Web site, can reference a global theme.

Theme Settings Precedence

- You can specify the precedence that theme settings take over local control settings by specifying how the theme is applied.
- If you set a page's Theme property, control settings in the theme and the page are merged to form the final settings for the control.
- If a control setting is defined in both the control and the theme, the control settings from the theme override any page settings on the control.
- This strategy enables the theme to create a consistent look across pages, even if controls on the pages already have individual property settings.

KAHE

- For example, it allows you to apply a theme to a page you created in an earlier version of ASP.NET.
- Alternatively, you can apply a theme as a style sheet theme by setting the page's StyleSheetTheme property.
- In this case, local page settings take precedence over those defined in the theme when the setting is defined in both places.
- This is the model used by cascading style sheets. You might apply a theme as a style sheet theme if you want to be able to set the properties of individual controls on the page while still applying a theme for an overall look.
- Global theme elements cannot be partially replaced by elements of application-level themes.
- If you create an application-level theme with the same name as a global theme, theme elements in the application-level theme will not override the global theme elements.

Properties You Can Define Using Themes

- As a rule, you can use themes to define properties that concern a page or control's appearance or static content.
- You can set only those properties that have a ThemeableAttribute attribute set to true in the control class.
- Properties that explicitly specify control behavior rather than appearance do not accept theme values. For example, you cannot set a Button control's CommandName property by using a theme.
- Similarly, you cannot use a theme to set a GridView control's AllowPaging property or DataSource property.

Themes vs. Cascading Style Sheets

Themes are similar to cascading style sheets in that both themes and style sheets define a set of common attributes that can be applied to any page. However, themes differ from style sheets in the following ways:

- 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. By default, any property values defined in a theme referenced by a page's Theme property override the property values declaratively set on a control, unless you explicitly apply the theme using the

KAHE

StyleSheetTheme property. For more information, see the Theme Settings Precedence section above.
- 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.

Security Considerations

Themes can cause security issues when they are used on your Web site. Malicious themes can be used to:

- Alter a control's behavior so that it does not behave as expected.
- Inject client-side script, therefore posing a cross-site scripting risk.
- Alter validation.
- Expose sensitive information.
- The mitigations for these common threats are:
- Protect the global and application theme directories with proper access control settings. Only trusted users should be allowed to write files to the theme directories.
- Do not use themes from an untrusted source. Always examine any themes from outside your organization for malicious code before using them on you Web site.
- Do not expose the theme name in query data. Malicious users could use this information to use themes that are unknown to the developer and thereby expose sensitive information.

Content Page Holder Control

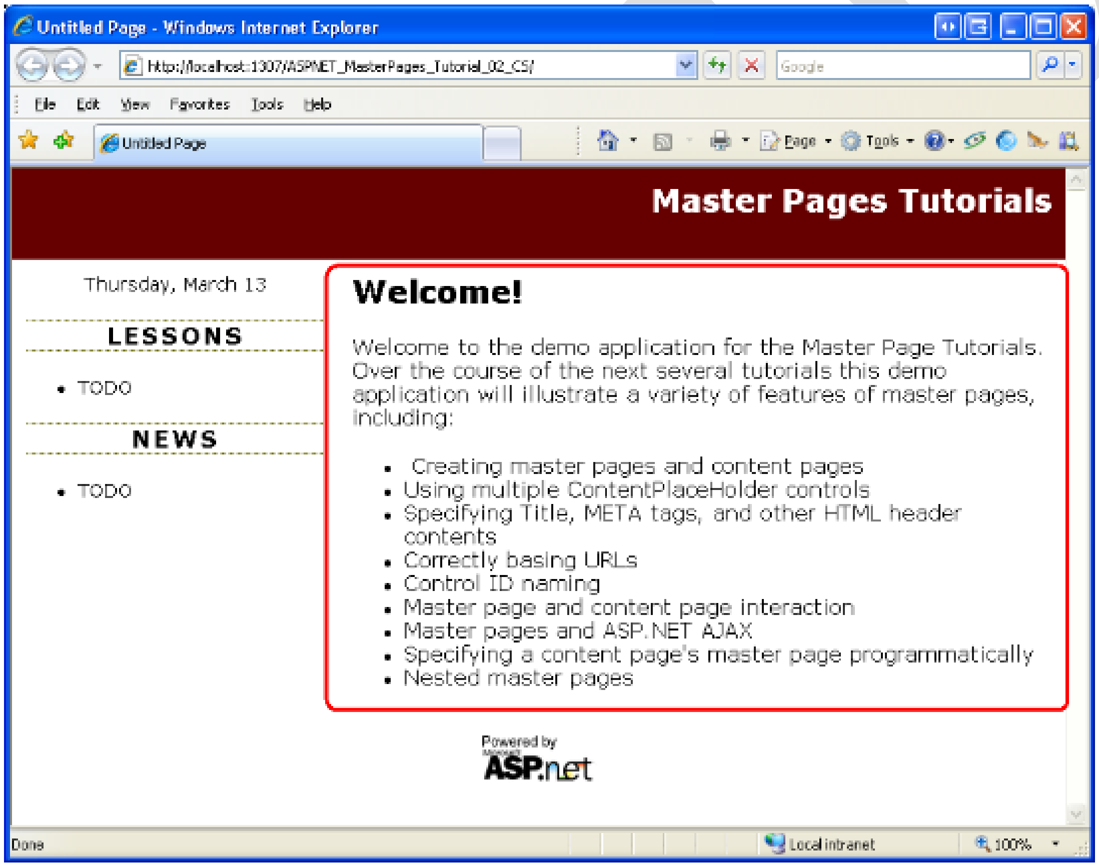
- In the preceding tutorial we examined how master pages enable ASP.NET developers to create a consistent site-wide layout.
- Master pages define both markup that is common to all of its content pages and regions that are customizable on a page-by-page basis.
- In the previous tutorial we created a simple master page (`Site.master`) and two content pages (`Default.aspx` and `About.aspx`).
- Our master page consisted of two ContentPlaceHolders named `head` and `MainContent`, which were located in the `<head>` element and Web Form, respectively.
- While the content pages each had two Content controls, we only specified markup for the one corresponding to `MainContent`.
- As evidenced by the two ContentPlaceHolder controls in `Site.master`, a master page may contain multiple ContentPlaceHolders.
- What's more, the master page may specify default markup for the ContentPlaceHolder controls.
- A content page, then, can optionally specify its own markup or use the default markup.

KAHE

- In this tutorial we look at using multiple content controls in the master page and see how to define default markup in the ContentPlaceHolder controls.

Step 1: Adding Additional ContentPlaceHolder Controls to the Master Page

- Many website designs contain several areas on the screen that are customized on a page-by-page basis. `Site.master`, the master page we created in the preceding tutorial, contains a single ContentPlaceHolder within the Web Form named `MainContent`. Specifically, this ContentPlaceHolder is located within the `mainContent <div>` element.
- Figure 1 shows `Default.aspx` when viewed through a browser.
- The region circled in red is the page-specific markup corresponding to `MainContent`.



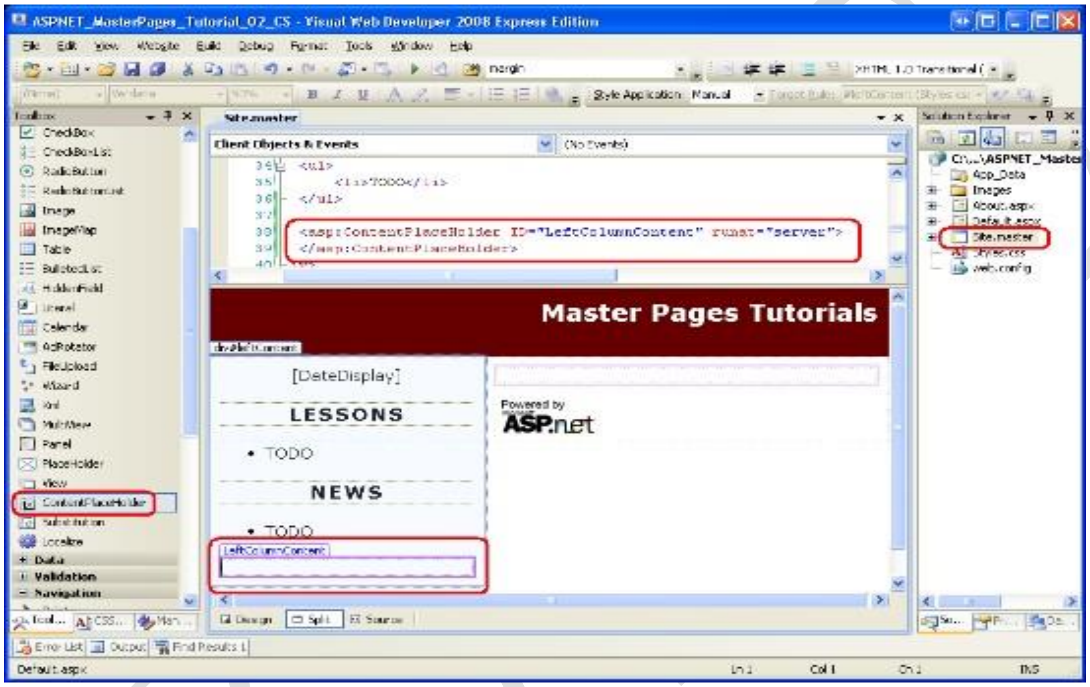
Welcome!

Welcome to the demo application for the Master Page Tutorials. Over the course of the next several tutorials this demo application will illustrate a variety of features of master pages, including:

- Creating master pages and content pages
- Using multiple ContentPlaceHolder controls
- Specifying Title, META tags, and other HTML header contents
- Correctly basing URLs
- Control ID naming
- Master page and content page interaction
- Master pages and ASP.NET AJAX
- Specifying a content page's master page programmatically
- Nested master pages

Figure 01: The Circled Region Shows the Area Currently Customizable on a Page-by-Page Basis (Click to view full-size image)

- Imagine that in addition to the region shown in Figure 1, we also need to add page-specific items to the left column beneath the Lessons and News sections.
- To accomplish this, we add another ContentPlaceHolder control to the master page.
- To follow along, open the `Site.master` master page in Visual Web Developer and then drag a ContentPlaceHolder control from the Toolbox onto the designer after the News section. Set the ContentPlaceHolder's `ID` to `LeftColumnContent`.



```
<ul>
    <li>TODO</li>
</ul>

<asp:ContentPlaceHolder ID="LeftColumnContent" runat="server">
</asp:ContentPlaceHolder>
```

Figure 02: Add a ContentPlaceHolder Control to the Master Page's Left Column (Click to view full-size image)

- With the addition of the `LeftColumnContent` ContentPlaceHolder to the master page, we can define content for this region on a page-by-page basis by including a Content control in the page whose `ContentPlaceHolderID` is set to `LeftColumnContent`.
- We examine this process in Step 2.

Step 2: Defining Content for the New ContentPlaceHolder in the Content Pages

- When adding a new content page to the website, Visual Web Developer automatically creates a Content control in the page for each ContentPlaceHolder in the selected master page.

- Having added a the `LeftColumnContent` ContentPlaceHolder to our master page in Step 1, new ASP.NET pages will now have three Content controls.
- To illustrate this, add a new content page to the root directory named `MultipleContentPlaceHolders.aspx` that is bound to the `Site.master` master page.

Visual Web Developer creates this page with the following declarative markup:

```
<%@ Page Language="C#" MasterPageFile="/Site.master" AutoEventWireup="true"
CodeFile="MultipleContentPlaceHolders.aspx.cs" Inherits="MultipleContentPlaceHolders"
Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="LeftColumnContent" Runat="Server">
</asp:Content>
```

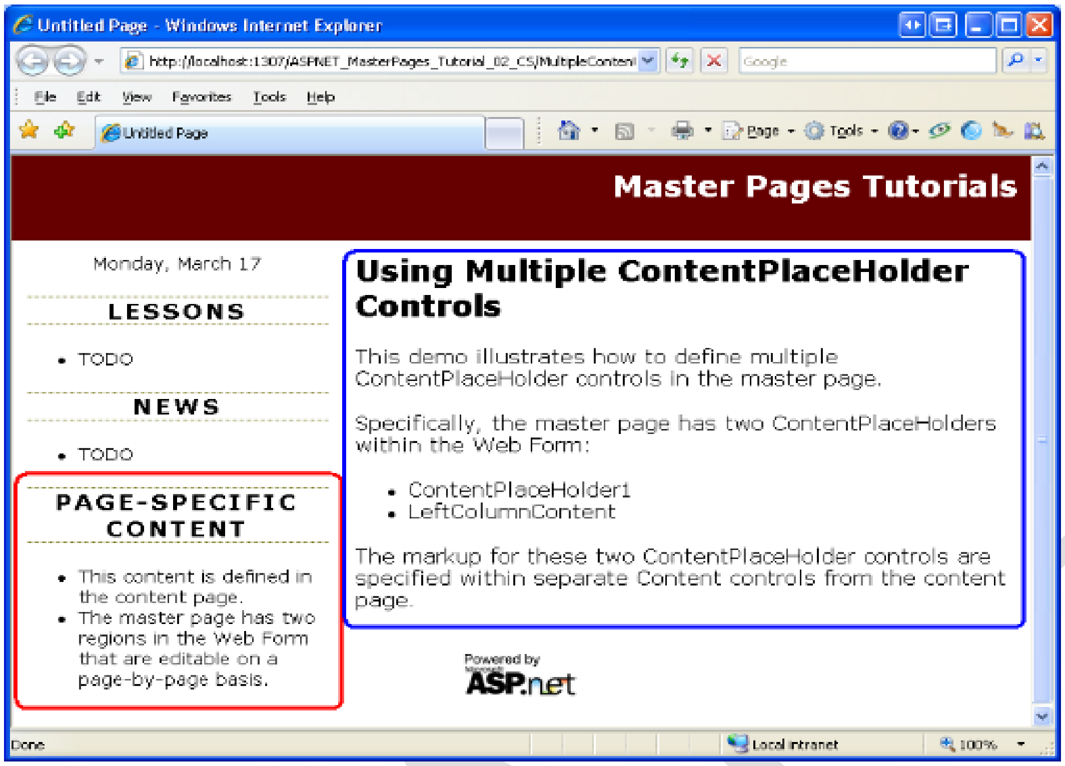
- Enter some content into the Content control referencing the `MainContent` ContentPlaceHolders (`Content2`).

Next, add the following markup to the `Content3` Content control (which references the `LeftColumnContent` ContentPlaceHolder):

```
<h3>Page-Specific Content</h3>
<ul>
 <li>This content is defined in the content page.</li>
 <li>The master page has two regions in the Web Form that are editable on a
 page-by-page basis.</li>
</ul>
```

- After adding this markup, visit the page through a browser.
- As Figure 3 shows, the markup placed in the `Content3` Content control is displayed in the left column beneath the News section (circled in red).
- The markup placed in `Content2` is displayed in the right portion of the page (circled in blue).

KAHE


Untitled Page - Windows Internet Explorer

Master Pages Tutorials

Monday, March 17

LESSONS
- TODO

NEWS
- TODO

PAGE-SPECIFIC CONTENT
- This content is defined in the content page.
- The master page has two regions in the Web Form that are editable on a page-by-page basis.

Using Multiple ContentPlaceHolder Controls

This demo illustrates how to define multiple ContentPlaceHolder controls in the master page.

Specifically, the master page has two ContentPlaceHolders within the Web Form:
- ContentPlaceHolder1
- LeftColumnContent

The markup for these two ContentPlaceHolder controls are specified within separate Content controls from the content page.

Powered by ASP.net

Figure 03: The Left Column Now Includes Page-Specific Content Beneath the News Section (Click to view full-size image)

Defining Content in Existing Content Pages

- Creating a new content page automatically incorporates the ContentPlaceHolder control we added in Step 1.
- But our two existing content pages - `About.aspx` and `Default.aspx` - don't have a Content control for the `LeftColumnContent` ContentPlaceHolder.
- To specify content for this ContentPlaceHolder in these two existing pages, we need to add a Content control ourselves.
- Unlike most ASP.NET Web controls, the Visual Web Developer Toolbox does not include a Content control item.
- We can manually type in the Content control's declarative markup into the Source view, but an easier and quicker approach is to use the Design view.
- Open the `About.aspx` page and switch to the Design view.
- As Figure 4 illustrates, the `LeftColumnContent` ContentPlaceHolder appears in the Design view; if you mouse over it, the title displayed reads: "LeftColumnContent

(Master)." The inclusion of "Master" in the title indicates that there is no Content control defined in the page for this ContentPlaceHolder.

- If there exists a Content control for the ContentPlaceHolder, as in the case for `MainContent`, the title will read: "*ContentPlaceHolderID* (Custom)."
- To add a Content control for the `LeftColumnContent` ContentPlaceHolder to `About.aspx`, expand the ContentPlaceHolder's smart tag and click the Create Custom Content link.

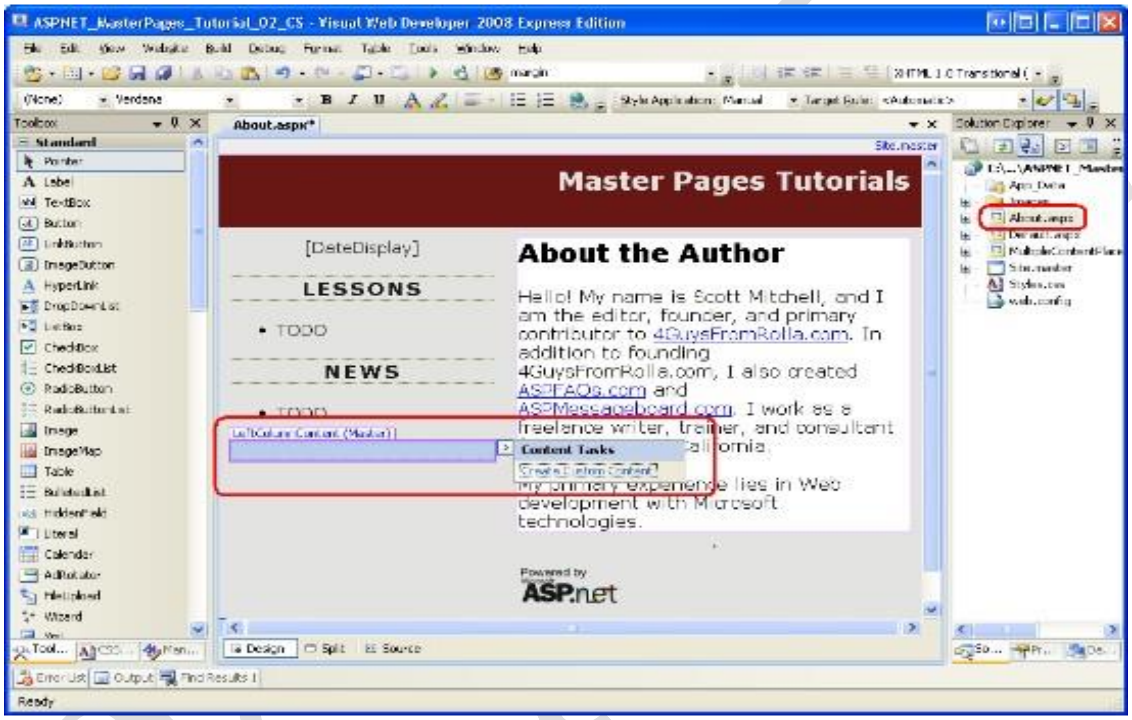


Figure 04: The Design View for `About.aspx` Shows the `LeftColumnContent` ContentPlaceHolder ([Click to view full-size image](#))

- Clicking the Create Custom Content link generates the necessary Content control in the page and sets its `ContentPlaceHolderID` property to the ContentPlaceHolder's `ID`.

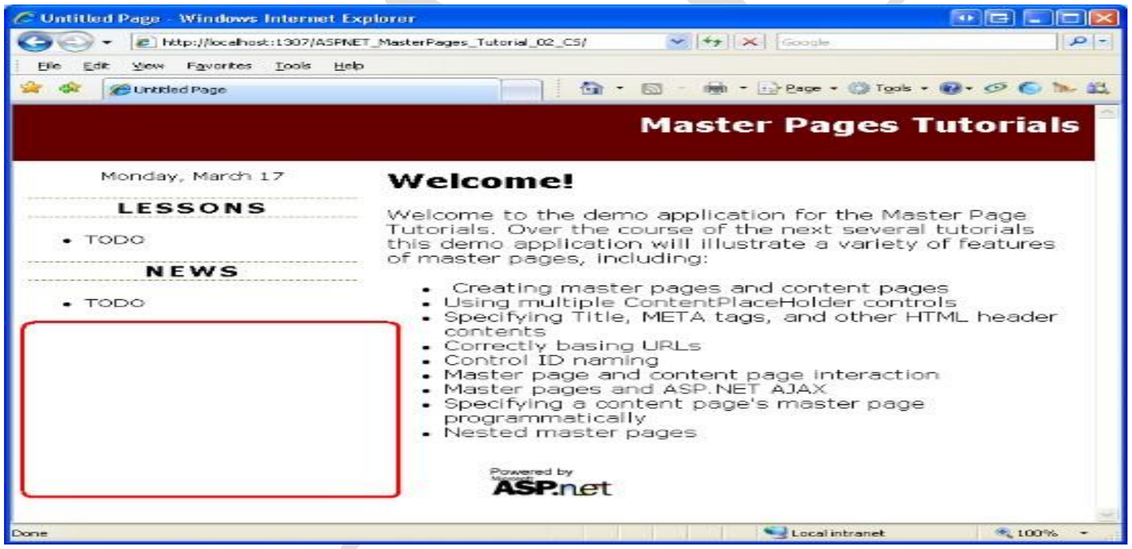
For example, clicking the Create Custom Content link for `LeftColumnContent` region in `About.aspx` adds the following declarative markup to the page:

```
<asp:Content ID="Content3" runat="server" contentplaceholderid="LeftColumnContent">

</asp:Content>
```

Omitting Content Controls

- ASP.NET does not require that all content pages include Content controls for each and every ContentPlaceHolder defined in the master page.
- If a Content control is omitted, the ASP.NET engine uses the markup defined within the ContentPlaceHolder in the master page.
- This markup is referred to as the ContentPlaceHolder's *default content* and is useful in scenarios where the content for some region is common among the majority of pages, but needs to be customized for a small number of pages.
- Step 3 explores specifying default content in the master page.
- Currently, `Default.aspx` contains two Content controls for the `head` and `MainContent` ContentPlaceHolders; it does not have a Content control for `LeftColumnContent`.
- Consequently, when `Default.aspx` is rendered the `LeftColumnContent` ContentPlaceHolder's default content is used.
- Because we have yet to define any default content for this ContentPlaceHolder, the net effect is that no markup is emitted for this region.
- To verify this behavior, visit `Default.aspx` through a browser.
- As Figure 5 shows, no markup is emitted in the left column beneath the News section.


Untitled Page - Windows Internet Explorer
http://localhost:1307/ASPNET_MasterPages_Tutorial_02_CS/
File Edit View Favorites Tools Help
Untitled Page
Master Pages Tutorials
Monday, March 17
LESSONS
TODO
NEWS
TODO
Powered by ASP.net
Done
Local intranet
100%

Welcome!

Welcome to the demo application for the Master Page Tutorials. Over the course of the next several tutorials this demo application will illustrate a variety of features of master pages, including:

- Creating master pages and content pages
- Using multiple ContentPlaceHolder controls
- Specifying Title, META tags, and other HTML header contents
- Correctly basing URLs
- Control ID naming
- Master page and content page interaction
- Master pages and ASP.NET AJAX
- Specifying a content page's master page programmatically
- Nested master pages

Figure 05: No Content is Rendered for the `LeftColumnContent` ContentPlaceHolder

Step 3: Specifying Default Content in the Master Page

- Some website designs include a region whose content is the same for all pages in the site except for one or two exceptions.
- Consider a website that supports user accounts.
- Such a site requires a login page where visitors can enter their credentials to sign into the site.
- To expedite the sign in process, the website designers might include username and password textboxes in the upper left corner of every page to allow users to sign in without having to explicitly visit the login page.
- While these username and password textboxes are helpful in most pages, they are redundant in the login page, which already contains textboxes for the user's credentials.
- To implement this design, you could create a ContentPlaceHolder control in the upper left corner of the master page.
- Each page that needed to display the username and password textboxes in their upper left corner would create a Content control for this ContentPlaceHolder and add the necessary interface.
- The login page, on the other hand, would either omit adding a Content control for this ContentPlaceHolder or would create a Content control with no markup defined.
- The downside of this approach is that we have to remember to add the username and password textboxes to every page we add to the site (except for the login page).
- This is asking for trouble.
- We're likely to forget to add these textboxes to a page or two or, worse, we might not implement the interface correctly (perhaps adding just one textbox instead of two).
- A better solution is to define the username and password textboxes as the ContentPlaceHolder's default content.
- By doing so, we only need to override this default content in those few pages that do not display the username and password textboxes (the login page, for instance).
- To illustrate specifying default content for a ContentPlaceHolder control, let's implement the scenario just discussed.

Adding a ContentPlaceHolder and Specifying Its Default Content

Open the `Site.master` master page and add the following markup to the left column between the `DateDisplay` Label and Lessons section:

```
<asp:ContentPlaceHolder ID="QuickLoginUI" runat="server">
 <asp:Login ID="QuickLogin" runat="server"
   TitleText="<h3>Sign In</h3>"
   FailureAction="RedirectToLoginPage">
 </asp:Login>
</asp:ContentPlaceHolder>
```

After adding this markup your master page's Design view should look similar to Figure 6.

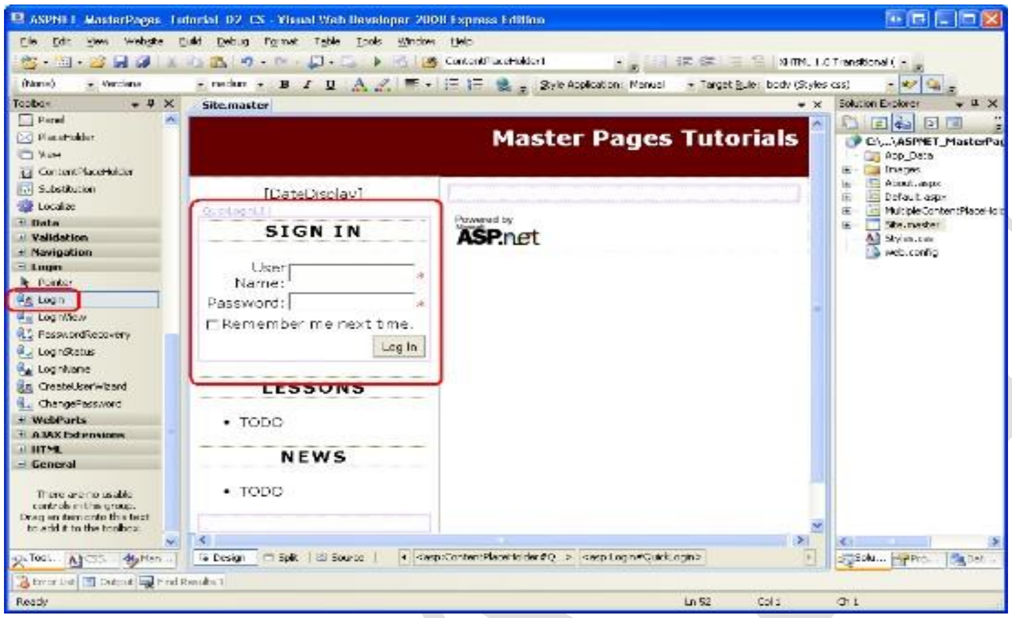


Figure 06: The Master Page Includes a Login Control (Click to view full-size image)

- This ContentPlaceHolder, `QuickLoginUI`, has a Login Web control as its default content. The Login control displays a user interface that prompts the user for their username and password along with a Log In button.
- Upon clicking the Log In button, the Login control internally validates the user's credentials against the Membership API.
- To use this Login control in practice, then, you need to configure your site to use Membership.
- This topic is beyond the scope of this tutorial; refer to my Forms Authentication, Authorization, User Accounts and Roles tutorials for more information on building a web application that supports user accounts.
- Feel free to customize the Login control's behavior or appearance.
- I have set two of its properties: `TitleText` and `FailureAction`.
- The `TitleText` property value, which defaults to "Log In", is displayed at the top of the control's user interface.
- I have set this property so that it displays the text "Sign In" as an `<h3>` element.

- The `FailureAction` property indicates what to do if the user's credentials are invalid.
- It defaults to a value of `Refresh`, which leaves the user on the same page and displays a failure message within the Login control.
- I've changed it to `RedirectToLoginPage`, which sends the user to the login page in the event of invalid credentials.
- I prefer to send the user to the login page when a user attempts to login from some other page, but fails, because the login page can contain additional instructions and options that would not easily fit into the left column.
- For example, the login page might include options to retrieve a forgotten password or to create a new account.

Creating the Login Page and Overriding the Default Content

- With the master page complete, our next step is to create the login page.
- Add an ASP.NET page to your site's root directory named `Login.aspx`, binding it to the `Site.master` master page.
- Doing so will create a page with four Content controls, one for each of the ContentPlaceHolders defined in `Site.master`.
- Add a Login control to the `MainContent` Content control.
- Likewise, feel free to add any content to the `LeftColumnContent` region.
- However, make sure to leave the Content control for the `QuickLoginUI` ContentPlaceHolder empty.
- This will ensure that the Login control does not appear in the left column of the login page.

After defining the content for the `MainContent` and `LeftColumnContent` regions, your login page's declarative markup should look similar to the following:

```
<%@ Page Language="C#" MasterPageFile="/Site.master" AutoEventWireup="true"
CodeFile="Login.aspx.cs" Inherits="Login" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
 <h2>
 Sign In</h2>
 <p>
 <asp:Login ID="Login1" runat="server" TitleText="">
```

KAHE

```
 </asp:Login>
 </p>
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="QuickLoginUI" Runat="Server">
</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="LeftColumnContent" Runat="Server">
 <h3>Sign In Tasks</h3>
 <ul>
 <li>Create a New Account</li>
 <li>Recover Forgotten Password</li>
 </ul>
 <p>TODO: Turn the above text into links...</p>
</asp:Content>
```

Figure 7 shows this page when viewed through a browser. Because this page specifies a Content control for the `QuickLoginUI` ContentPlaceHolder, it overrides the default content specified in the master page. The net effect is that the Login control displayed in the master page's Design view (see Figure 6) is not rendered in this page.

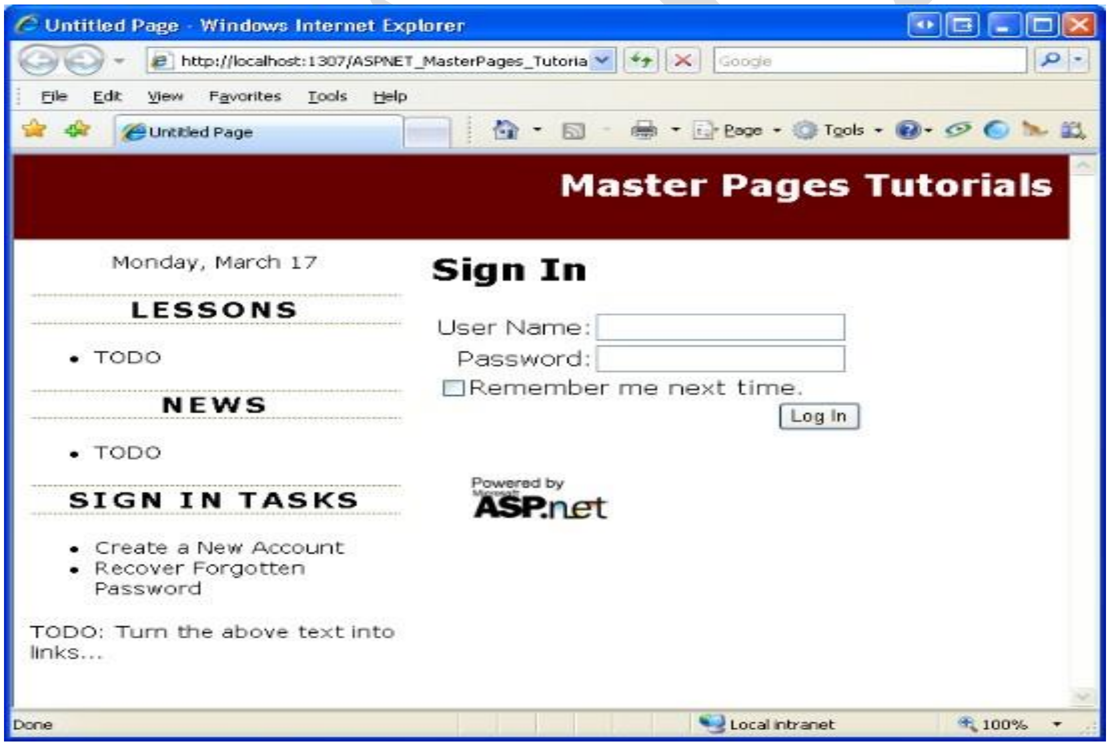


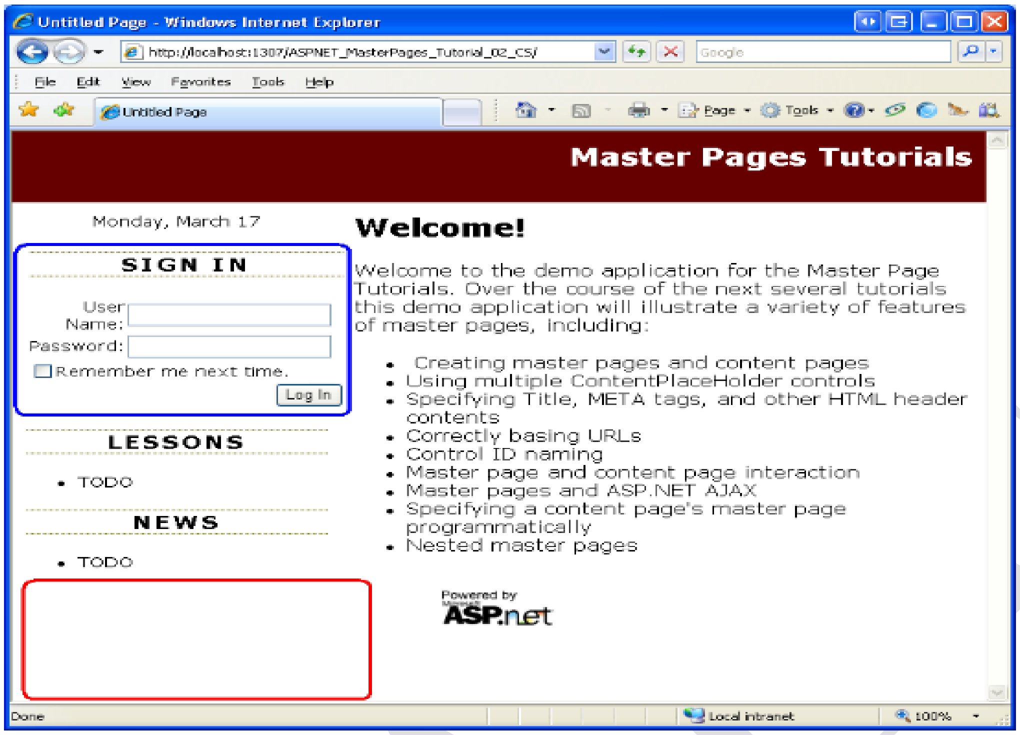
Figure 07: The Login Page Represses the `QuickLoginUI` ContentPlaceHolder's Default Content (Click to view full-size image)

Using the Default Content in New Pages

- We want to show the Login control in the left column for all pages except the Login page.
- To achieve this, all the content pages except for the login page should omit a Content control for the `QuickLoginUI` ContentPlaceHolder.
- By omitting a Content control, the ContentPlaceHolder's default content will be used instead.
- Our existing content pages - `Default.aspx`, `About.aspx`, and `MultipleContentPlaceHolders.aspx` - do not include a Content control for `QuickLoginUI` because they were created before we added that ContentPlaceHolder control to the master page.
- Therefore, these existing pages do not need to be updated. However, new pages added to the website include a Content control for the `QuickLoginUI` ContentPlaceHolder, by default.
- Therefore, we have to remember to remove these Content controls each time we add a new content page (unless we want to override the ContentPlaceHolder's default content, as in the case of the login page).
- To remove the Content control, you can either manually delete its declarative markup from the Source view or, from the Design view, choose the Default to Master's Content link from its smart tag.
- Either approach removes the Content control from the page and produces the same net effect.

Figure 8 shows `Default.aspx` when viewed through a browser. Recall that `Default.aspx` only has two Content controls specified in its declarative markup - one for `head` and one for `MainContent`. As a result, the default content for the `LeftColumnContent` and `QuickLoginUI` ContentPlaceHolders are displayed.

KAHE


Untitled Page - Windows Internet Explorer
http://localhost:1307/ASPNET_MasterPages_Tutorial_02_CS/
File Edit View Favorites Tools Help
Untitled Page
Page Tools
Done
Local intranet
100%

Master Pages Tutorials

Monday, March 17

SIGN IN

User Name:

Password:

Remember me next time.

Log In

LESSONS

- TODO

NEWS

- TODO

Welcome!

Welcome to the demo application for the Master Page Tutorials. Over the course of the next several tutorials this demo application will illustrate a variety of features of master pages, including:

- Creating master pages and content pages
- Using multiple ContentPlaceHolder controls
- Specifying Title, META tags, and other HTML header contents
- Correctly basing URLs
- Control ID naming
- Master page and content page interaction
- Master pages and ASP.NET AJAX
- Specifying a content page's master page programmatically
- Nested master pages

Powered by ASP.net

Figure 08: The Default Content for the `LeftColumnContent` and `QuickLoginUI` ContentPlaceHolders are Displayed.