Summary for exam 70-315 Developing Web applications using .NET

1. Introduction to Web Programming

1.1. There are four types of Internet applications: Web applications, Web services, Internet-enabled applications, and peer-to-peer applications

1.2. Web applications run on a server, processing user requests for pages and composing those pages using executable code and static resources on the server

1.3. Web applications can provide dynamic content based on dynamic server resources, such as a database, and based on user inputs, such as creating a mortgage payoff table from user loan information

1.4. ASP.NET is a platform for creating Web applications that run on Windows servers using IIS and the .NET Framework. It is made of the following components:

- Visual Studio .NET Web development tools
- The System.Web namespaces. Include the programming classes that deal with Web-specific items such as HTTP requests and responses, browsers, and e-mail.
- Server and HTML controls. These are the user-interface components that you use to gather information from and provide responses to users.

1.5. ASP.NET also uses other items such as:

- Microsoft Internet Information Services (IIS)
- The Microsoft Visual Basic .NET, Microsoft Visual C#, and JScript programming languages
- The .NET Framework
- ADO.NET database classes and tools

1.6. Web applications are made up of content, an executable, and configuration files

- Content. Web Forms, HTML, images, audio, video, other data
- Program logic. Executable files, scripts
- Configuration. Web configuration file, Style sheets, IIS settings

1.7. The content of a Web application is presented through Web forms. Web forms use HTML components like conventional HTML pages; like Windows forms, however, they can also respond to user events such as mouse clicks. Components in a Webform:

- Server controls. Run processes in the server and can save data between page displays: TextBox, Label, Button, ListBox, DropDownList, DataGrid
- HTML controls. Utilized when server controls are not needed: Text area, Table, Image, Submit Button, Reset Button.
- Data controls. To implement database and XML access: SqlConnection, SqlCommand, OleDbConnection, OleDbCommand, DataSet
- System components. To access system-level events in the server: FileSystemWatcher, EventLog, MessageQueue

1.8. The Web application' s executable is stored in a .dll file called an assembly. Assemblies are compiled to an intermediate state, and the final compilation is done by the CLR just before running the application.

1.9. The .NET Framework is made up of the CLR and the .NET class library. The .NET class library makes the CLR run-time tasks available to programmers.

1.10. The .NET classes are grouped by programming task into namespaces. These groupings help you locate the class, method, or property you need to accomplish a task

1.11. Use the Visual Studio .NET Start page to view current product information, to open new or existing projects, to set user environment preferences, and to sign up for Web hosting services

1.12. Edit Web Forms pages and HTML pages visually by using the Document window' s Design mode; edit them as text by using the Document window' s HTML mode

1.13. Set the Help language filter to view code samples in a single programming language or in multiple languages

1.14. Modify the Visual Studio .NET environment features using the Options dialog box

2. Creating Web Forms applications

2.1. Web applications use Web forms to create a user interface that is presented through an Internet browser on the user' s computer.

2.2. The code and resources that respond to events and perform useful tasks reside and run on the Web server hosting the application.

2.3. Because Web applications are distributed between a client and a server, there are four significant differences between programming for the Web and programming for Windows:

- Web applications use server controls and HTML controls rather than Windows controls.
- Web applications are displayed in a Web browser rather than in their own window.
- Web forms are not persistent while displayed. You must preserve persistent data in a state variable during page and control events.
- Processing occurs on the server and data is exchanged through a cycle of requests and responses.

2.4. Web applications are event-driven, and events occur at the application, page, and server control levels. Application and

session events are defined in the Globla.asax file.

Application event handlers

Event	When it occurs
Application_Start	The first user visits an application
Application_End	There are no more users of the application
Application_BeginRequest	Occurs every time a browser navigates to any of the pages in the application
Application_EndRequest	At the end of each request to the server
Session_Start	A new user visits the start page of the app
Session_End	A user leaves the application (by closing the brower or timing out)

Note: Session and Response objects are not available in Application_Start

2.5. There are two kinds of variables:

- Application variables. Availables to all users of an applicaton
- Session variables. Availables only to a single session

2.6. Web forms events**.** You use Web form events to process and maintain data used on a Web page, to respond to data binding, and to handle exceptions on the Web page. They occur in the following order:

Page_Init	Get server controls from the Web form' s view state
Page_Load	The server controls are loaded on the Page object. View state information is available at this point
Page_PreRender	The application is about to render the Page object
Page_Unload	The page is unloaded from memory
Page_Error	An unhandled exception occurs
Page_AbortTransaction	A transaction is aborted
Page_CommitTransaction	A transaction is accepted
Page_DataBinding	A server control on the page binds to a data source
Page_Disposed	The Page object is released from memory. This is the last event in the life of a Page object

2.7. Use the *IsPostBack* poperty in the *Page_Load* event to initialize data. This is similar to the *Application_Start* event but at page level.

2.8. There are three types of server controls events, which occur in the following order:

Validation events	Occur without the page being returned to the server. The validation server controls use these types of events
Cached events	They are collected while the page is displayed, and then processed once the page sends a request to the server
Post-back events	They cause the page to send a request to the server, but their event procedure is processed last in order of events handled

2.9. The events in the server occur in the following sequence. Check Pg 152:

1)	Validation Controls	
2)	*Page_Init*	
3)	*Page_load*	
4)	Cached events	Example: new text in a textbox triggers the TextChanged event
5)	Post-back events	A button was clicked causing this post-back
6)	*Page_PreRender*	
7)	*Page_unload*	

2.10. The boundaries of a Web application are determined by its folder structure. The application boundary starts in the folder containing the start page of the application and it ends at the last subordinate folder

2.11. Application boundaries affect the scope of data stored in Application state and they allow you to specify the process in which the server runs your application. It' s important to realize that all of the DLLs within an application boundary have access to the same Application state

2.12. You use IIS to create root folders for your applications, set application boundaries, and determine the process in which IIS runs your application.

2.13. You have only indirect control over when an application ends through the session timeout. When the last session ends, IIS ends the application

3. Working with Web Objects

3.1. Namespaces organize your code and provide access to code in the .NET Framework. To use a namespace from outside of your project, establish a reference to it using the References dialog box from the Project menu. Add an *using* statement to the source file to provide a shortcut for referring to members of the namespace in code.

3.2. Classes define objects within a namespace. You can base one class on another using inheritance. When using inheritance, the base class provides its members to the derived class where they can be overridden, overloaded, or hidden. Visual Studio does not support deriving new Web forms from existing base Web forms.

3.3. Web applications use the *System.Web* and *System.Web.UI namespaces*. These namespaces define most of the objects used in a Web application, including the *Application, Page, Request*, and *Response* objects. These four objects provide access to most of the subordinate objects in a Web application.

- Application object. Derives from the *HttpApplication* class and resides in the Global.asax file with the name Global
- Web form object. Derives from the *Page* object and resides in the Web Form classes

3.4. These *Global* and *WebForm1* objects are the entry points you use to get at other Web objects in an application. Those objects, along with *Request* and *Response*, are the objects you will most commonly work with in code.

3.5. Because ordinary variables defined in a Web form are not persistent, you need to save volatile data between requests. ASP.NET lets you save data items as

1. Query strings
2. Cookies
3. ViewState
4. Session state
5. Application state.

3.5.1. Query Strings

```
***Using an hyperlink.
<A HREF= "WebForm1.aspx?UName=Wombat">Query string sample.
</A>
***Implemented in code
private void Button1_Click(object sender,
                                  System.EventArgs e)
{
    // Redisplay this page with a QueryString
    Response.Redirect("Webform1.aspx?UName=Wombat");
}
***Reading a Querystring
private void Page_Load(object sender, System.EventArgs e)
{
    // Get and display the query string.
    Response.Write(Request.QueryString["UName"]);
}
```

3.5.2. Cookies

```
***Setting a cookie
private void Page_Load(object sender, System.EventArgs e)
{
  // Run the first time this page is displayed.
  if(!IsPostBack)
    // If the browser supports cookies.
    if(Request.Browser.Cookies)
    {
      // Create a cookie.
      HttpCookie cookUPrefs = new HttpCookie("UPrefs");
      cookUPrefs.Value = "English";
      // Add the cookie.
      Response.Cookies.Add(cookUPrefs);
    }
}

***Reading a cookie
private void Page_Load(object sender, System.EventArgs e)
{
   // Run the first time this page is displayed.
    if(!IsPostBack)
      // If the browser supports cookies.
      if(Request.Browser.Cookies)
        // Check if the UPrefs cookie exists
```

```
        if(Request.Cookies["UPrefs"] != null)
          // Save the value of the cookie.
         Session["Lang"] = Request.Cookies["UPrefs"].Value;
}
```

3.5.3. ViewState.

ViewState is optimized for String, ArrayList, and HashTable data types, but can save any data that can be serialized or that provides a TypeConverter. For other types of data, you can create your own code to save and restore ViewState by overriding the SaveViewState and LoadViewState methods.

```
private void Button1_Click(object sender,
                                System.EventArgs e)
{
      // Add text to the view state.
    ViewState.Add(ViewState.Count.ToString(), TextBox1.Text);
}

private void Page_Load(object sender, System.EventArgs e)
{
    Int ICount = 0;
    if (IsPostBack)
        // For each item in the ViewState
        foreach(StateItem staItem in ViewState.Values)
        {
            //Populating a table
            TableRow rowNew = new TableRow();
            TableCell celNew = new TableCell();
            celNew.Text = staItem.Value.ToString();       // Set cell text.
            rowNew.Cells.Add(celNew);                     // Add cell to row.
            Table1.Rows.Add(rowNew);                      // Add row to table

            //Simple displaying
            Icount ++;
            myKey = "Key" +  Icount.ToString();
            Response.Write(ViewState[myKey]);
            Response.Write(staItem.Value.ToString())
        }
}
```

3.5.4. Session and Session state

3.5.4.1. There is not type checking in Application and Session state. The best is to store their values into page-level variables in the *Page_load* event procedure. First check if they are null and then assign them to a variable of the datatype they should belong to.

3.5.4.2. Session state affects performance and it can be turned off. To do it change the following in the Web.config file.

```
<sessionstate mode="False". . .>

//Using Session to keep a dataset
     // Check if this is the first time page is displayed.
     if (!IsPostBack)
     {
           // On first display:
           //   Fill the data set.
           adptContacts.Fill(dsContacts);
           //   Save data set as state variable.
           Session["dsContacts"] = dsContacts;
     }
     else
           // On subsequent displays:
           //   Get the data set from the state variable.
           dsContacts = (dsContacts)Session["dsContacts"];
     // Bind to data set.
     grdContacts.DataBind();
```

3.5.4.3. Application state are available only throughout the current process, but not between multiple processors or multiple servers.

```
string mstrUname = "";

private void Page_Load(object sender,
                           System.EventArgs e)
{
    // Check if state variable exists.
    if(Application["Uname"] != null)
        // Get state variable.
        mstrUname = Application["Uname"].ToString();
```

```
    // Set variable
    mstrUname = "Wombat";
    // Use variable.
    Response.Write(mstrUname);
}

private void Page_UnLoad(object sender,
                         System.EventArgs e)
{
    // Save the state variables back.
    Application["Uname"] = mstrUname;
}
```

4. Using Controls

4.1. Web Form layout.

Arrangement of controls in a web form can be either Grid or Flow layout. In Grid layout controls are placed exactly where they are dragged, in Flow layout the controls are place one after the other, like when adding them at run time (useful when mixing controls and text).

4.2. Controls utilized in a Web Form.

A form can have HTML and Server controls. Server controls provide properties, methods, and events that can be used in server code.

4.2.1. Advantages of Server controls over HTML controls:

Server events	HTML only triggers page-level post-back events in the server
State management	Data entered in a control is maintained across requests
Adaptation	Automatically detects and adapts to the browser
Properties	NET specific properties

4.2.2. For data binding is better to use HTML controls or server controls with state management off (*EnableViewState = false*).

4.2.3. HTML controls provide attributes and events that can be used in scripts that run on the client. To access properties of an HTML control in server code, add a *runat=server* attribute to the control' s HTML definition.

4.2.4. Server Controls vs. HTML Controls.

Task	Server controls	HTML controls
Display text	Label, TextBox, Literal	Label, Text Field, Text Area, Password Field
Display tables	Table, DataGrid	Table
Select from list	DropDownList, ListBox, DataList, Repeater	List Box, Dropdown
Perform commands	Button, LinkButton, ImageButton	Button, Reset Button, Submit Button
Set values	CheckBox, CheckBoxList, RadioButton, RadioButtonList	Checkbox, Radio Button
Display images	Image, ImageButton	Image
Navigation	Hyperlink	none (use <a> tags in text)
Group controls	Panel, Placeholder	Flow Layout, Grid Layout
Work with dates	Calendar	none
Display ads	AdRotator	none
Display horizontal rules	Literal	Horizontal Rule

Task	Server controls	HTML controls
Get filenames from client	none	File Field
Store data on page	(provided by state management)	Input Hidden
Validate data	RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, CustomValidator,ValidationSummary	none (use page-level)

4.3. Working with text

4.3.1. Options to display text

- Response.Write("Some text") .
- Label control
- Read Only TextBox
- Literal Control

4.4. Working with tables and lists

4.4.1. ASP.NET List and Table Controls

ListBox	Display read-only text in a simple scrollable list format.
DropDownList	Display read-only text in a simple drop-down list format.
Table	Use TableRows and TableCells collections.
DataGrid	Display text and controls in columns and rows using a template to control appearance. DataGrid controls have built-in formatting, sorting, and paging capabilities.
DataList	Display rows of text and controls using a template to control appearance. DataList controls have built-in formatting and selection capabilities.
Repeater	Display rows of other controls using a template to control appearance. Repeater controls do not include the built-in capabilities found in the DataGrid and DataList controls.

4.4.2. Adding items to Lists or tables:

4.4.2.1. At design time use the Collection Editor dialog box.

4.4.2.2. At run time, using the *Add* method of the control's *Items* collection.

```
ListBox1.Items.Add(txtSource.Text);
DropDownList1.Items.Add(txtSource.Text);
```

For a Table control will automatically store data only for the table cells created at design time in the Collection Editor

4.4.2.3. To create additional Rows and cells in a table, use the TableRow and TableCells collections:

```
 string[] arrWords;
string strWords;
TableRow rowNew;
TableCell celNew;
// For each string saved in ViewState.
for (int iCount1 = 0; iCount1 < ViewState.Count;
        iCount1++)
{
    // Create a new table row.
    rowNew = new TableRow();
    // Get the string from ViewState.
    strWords = ViewState(iCount1);
    // Break comma separated value into an array.
    arrWords = Split(strWords, ",");
    // For each item in the array.
    for (int iCount2 = 0;
       iCount <= arrWords.GetUpperBound; iCount2++)
    {
        // Create a new table cell.
        celNew = new TableCell();
```

```
            // Set the text to display in the cell.
            celNew.Text = arrWords[iCount2];
            // Add the cell to the table row.
            rowNew.Cells.Add(celNew);
        }
        // Add the row to the table.
        Table1.Rows.Add(rowNew);
```

4.4.3. Use the SelecteItem property to get the current selection.

```
        // Test is an item is selected.
if (ListBox1.SelectedItem == null)
  Label1.Text = "No item is selected.";
else
  // Display the selected item.
  Label1.Text = "Selected:" + ListBox1.SelectedItem.Text;
```

4.4.4. Using Simple Data Binding with Lists:

4.4.4.1. Any public data can a be source for simple data binding. Here a string is utilized to populate a DropDowList.

4.4.4.1.1. Set the code

```
public string[] arrData= {"This", "that", "and", "the",  "other"};

private void Page_Load(object sender, System.EventArgs e)
{
    Page.DataBind()
}
```

4.4.4.1.2. Using Studio chose the DataBinding property, then chose DataSource as the property to bind. Set as "Custom Binding Expression" the string array.

4.4.4.2. When using *Page.DataBind(),* all the controls in the page with bindings are processed. (Pg. 147).

4.4.5. Adding Items to DataGrid. DataList, and Repeater Controls.

4.4.5.1. Use data binding to add items and Templates to display them.

4.4.5.2. A Template is set of HTML elements or/and Server controls that will be repeated for each data item in the control.

4.4.5.3. Use Studio to define the controls that make the template and binding them to the source.

```
//How to define an item on a datalist with a Dataset as a datasource (Studio generated)
<asp:Label id="lblDate" runat="server"
   Text='<%# DataBinder.Eval(Container, "DataItem.CallDate", "{0:d}") %>'>
```

4.5. Performing Commands.

4.5.1. Controls that trigger post-back events are Button, LinkButton, and ImageButton.

4.6. Getting and Setting values:

To get Boolean values from the user use RadioButtons, RadioButtonList, or CheckBoxList controls.

4.6.1. Use the Checked property to get the setting from the CheckBox and RadioButton.

```
Response.Write("Checkbox1 is " + CheckBox1.Checked.ToString());
```

4.6.2. RadioButton controls get related when they have the same *GroupName* property.

4.6.3. The values of CheckBoxList and RadioButtionList controls are set in ListItem items. To browse them use a *Foreach* statement.

```
foreach (ListItem lstItem in RadioButtonList1.Items)
    {
        if (lstItem.Selected)
            Response.Write(lstItem.Text + " is selected.<br>");
    }
```

4.7. Displaying Graphics and Advertisements

4.7.1. Ways to display graphics:

4.7.1.1. As a Background. Using the WebForm's properties Background and BackImageUrl.

4.7.1.2. As a Foreground. Using the Image control.

4.7.1.3. As a Button. Using the ImageButton.

4.7.1.4. As and Ad. Using the AdRotator.

4.7.2. The Image controls are utilized only to display images at run time:

```
Image1.ImageUrl = "gondola.jpg";
```

4.8. Grouping Controls.

4.8.1. Use the Panel controls to group controls. Here a panel is hidden when condition is given.

```
if ((txtName.Text == "Guest") && (txtPassword.Text == "Wombat"))
    {
        pnlLogon.Visible = False;
        lblWelcome.Text = "Welcome " + txtName.Text;
    }
else
```

```
    {
        txtPassword.Text = "";
        vldPassword.Validate();
    }
```

4.9. Getting Dates

Use the Calendar control de get or display dates. Use the postback *SelectionChanged* event procedure and the *SelectedDate* or *SelectedDates* properties to get the date.

```
private void calSource_SelectionChanged(object sender,
      System.EventArgs e)
{
    // Display the current date.
    lblDate.Text = "Current date: " + calSource.TodaysDate;
    if (calSource.SelectedDates.Count == 1)
        // If one date is selected, display it.
        lblDate.Text = "Selected date: " + calSource.SelectedDate;
    else
        // If multiple dates are selected, display them.
        lblDate.Text = "Selected dates: " +
                  calSource.SelectedDates[0] + " to " +
                  calSource.SelectedDates [calSource.SelectedDates.Count - 1];

}
```

4.10. Getting files from the Client.

4.10.1. Use the "File Filed" HTML control to upload files from the client to the server.

4.10.2. Paths handling.

```
//Get the path to the virtual directory
 string myVirtualPath = Request.ApplicationPath
//Map the specified virtual path to a physical path
String MyPhysicalPath = Request.MapPath(myVirtualPath)
//Using them together to get the Virtual directory's physical path.
String MyPhysicalPath = Request.MapPath(Request.ApplicationPath)
```

4.10.3. To receive files from the client you have to set "Run as Server Control" for the form, add the following attributes to the *Form* tag:

```
<form action="webform1.aspx" method="post" enctype="multipart/form-data" ….>
```

And include the following code

```
protected System.Web.UI.HtmlControls.HtmlInputFile filUpload; //FileFiled HTML as a server
cont.
protected System.Web.UI.WebControls.Button butUpload; //Button to read the chosen file.
.
private void butUpload_Click(object sender,
      System.EventArgs e)
{
    // Using "File Filed" properties to get the full file name
    string strFilename = filUpload.PostedFile.FileName;
    // Get the base name.
    strFilename =  System.IO.Path.GetFileName(strFilename);
    // Save file as base name on server.
    filUpload.PostedFile.SaveAs(Request.MapPath(Request.ApplicationPath) +  "\\" +
                                strFilename);
  }
```

4.11. Validating Data

4.11.1. Validation Controls validate before the page is posted. They use a Jscript library named WebUIValidation.js that is downloaded to the Client and is compatible only with explorer 4.0 or newer. Validation Controls also perform Server validation.

4.11.2. ASP.NET performs control validation on the client side immediately before posting the Web form back to the server. Once client-side validation succeeds, the Web form is validated again on the server side before the Page_Load event occurs. This ensures that the Web form has not been modified on the client to bypass validation and it also guarantees that validation will work with pre-Internet Explorer 4.0 browsers.

4.11.3. ASP.NET Validation Controls

Validation control	Use to
RequiredFieldValidator	Check if a control contains data
CompareValidator	Check if an entered item matches an entry in another control
RangeValidator	Check if an entered item is between two values
RegularExpressionValidator	Check if an entered item matches a specified format
CustomValidator	Check the validity of an entered item using a client-side script or a server-side code, or both
ValidationSummary	Display validation errors in a central location or display a general validation error description

4.11.4. Properties to use in Validation Controls:

4.11.4.1. *ControlToValidate*. The control to validate

4.11.4.2. *ControlToCompare*. When a comparison of two controls is performed

4.11.4.3. *ErrorMessage*. Message to display in case of error.

4.11.4.4. *Text*. Used instead of *ErrorMessage* property. To display where the error occurred and to display the *ErrorMessage* property in a *ValidationSummary* control.

4.11.5. Combining Validations.

4.11.5.1. All controls take blanks as a valid entry –excepting *RequieredFieldValidator*-, combining controls you can validate blanks or values out or range.

4.11.5.2. Use multiple validation controls to check multiple conditions on a single data field. For instance, a TextBox that requires a telephone number should be validated by both a RequiredFieldValidator and a RegularExpressionValidator control

4.11.6. Canceling validation.

Provide a Cancel button to prevent a user to become trapped by validation rules:

4.11.6.1. Create and Submit HTML button with the property:

```
<INPUT id="butCancel" type="reset" value="Cancel" onclick="Page_ValidationActive=false;
    window.navigate('Switchboard.aspx'); " name="butCancel" runat="server">
```

4.11.6.2. Because the *Page.IsValid* property is set to True, you need to validate again the page in the *Page_Load* event:

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Validate in case user cancelled validation.
    if (Page.IsPostBack)
    {
        Page.Validate();
        if (!Page.IsValid)
            // User cancelled validation.
            Response.Redirect("default.htm");
    }
}
```

4.11.7. Customizing Validation

4.11.7.1. Use the CustomValidator control and write code in to perform Server side validation and optionally Client side.

4.11.7.2. For Server side place the validation in the ServerValidate event process and use the arguments to access the controls to validate.

```
//Double-clicking the Custom-validator controls creates this event handler
private void vldtxtPrime_ServerValidate(object source,
      System.Web.UI.WebControls.ServerValidateEventArgs args)
{
    try
    {
        // Get value from ControlToValidate (passed as args).
        int iPrime = Int32.Parse(args.Value);
            // Validation routine
        for (int iCount = 2; iCount <= (iPrime / 2); Count++)
        {
            // If number is evenly divisible, it's not prime,return False.
            if((iPrime % iCount) == 0)
            {
                args.IsValid = false;
                return;
            }
            // Number is prime, return True.
            args.IsValid = true;
            return;
        }
    }
    catch(Exception e)
    {
        // If there was an error parsing, return False.
        args.IsValid = false;
        return;
    }
}
```

4.11.7.3. For Client side specify a validation script in the CustomValidator control' s *ClientValidationFunction* property. It is optional

```
<script language="jscript">
        function ClientValidate(source, arguments)
        {
            for (var iCount = 2; iCount <= arguments.Value / 2; iCount++)
            {
                // If number is evenly divisible, it's not prime. Return false.
```

```
                if ((arguments.Value % iCount) == 0)
                {
                    arguments.IsValid = false;
                    return false;
                }
            }
            // Number is prime, return True.
            arguments.IsValid = true;
            return true;
        }
</script>

<asp:CustomValidator id="vldtxtPrime" runat="server" ErrorMessage="CustomValidator"
 ControlToValidate="txtPrime" ClientValidationFunction="ClientValidate">
 </asp:CustomValidator></P>
```

4.11.7.4. Navigating Between forms

4.11.7.4.1. Navigating Between Pages

Hyperlink control	Navigate to another page.
Response.Redirect method	Navigate to another page from code. This is equivalent to clicking a hyperlink.
Server.Transfer method	End the current Web form and begin executing a new Web form. This method works only when navigating to a Web Forms page **(.aspx).**
Server.Execute method	Begin executing a new Web form while still displaying the current Web form. The contents of both forms are combined. This method works only when navigating to a Web Forms page **(.aspx).**
Window.Open script method	Display a page in a new browser window on the client.

4.11.7.5. Hyperlink server controls don't have server-side user events, just display the page defined in the *NavigateURL* property.

4.11.7.6. To navigate using the *LinkButton* or *ImageButton* use the *Response.Redirect* method.

4.11.7.7. *Transfer* can retain some information from the source page across requests. Setting the Transfer method' s *preserveForm* argument to True makes the form' s *QueryString, ViewState*, and event procedure information available in the destination form. To do it set the *EnableViewStateMac* attribute of the form to False (To avoid ViewState hashing…)

```
<%@ Page language="vb" EnableViewStateMac="false" . . .>

//Transferring  to a new form passing information

// Webform1.aspx
private void ImageButton1_Click(object sender,
    System.Web.UI.ImageClickEventArgs e)
{
    // Transfer to another form, retaining ViewState.
    Server.Transfer("Webform2.aspx", true);
}

//Use the Request object to retrieve the viewState information from the source form

// Webform2.aspx
    private void Page_Load(object sender, System.EventArgs e)
    {
       System.Collections.Specialized.NameValueCollection colForm;
       // Get data from the source Web form.
       colForm = Request.Form;
       // Display the value from Webform1's TextBox.
       Response.Write("TextBox1.Text: " + colForm["TextBox1"] + "<br>");
       // Display the X, Y coordinated of where the click occurred.
       Response.Write("ImageButton X, Y coords: " +
             colForm["imgTransfer.x"] + ", " +
             colForm["imgTransfer.y"] + "<br>");
    }
```

4.11.7.8. Using the Execute method

4.11.7.8.1. Directs the results from a Web form to a region on the current page. Because any post-back event in the second form will clear the first form it is recommendable to use it when the second form doesn't trigger post-back events. It requires the *EnableViewStateMac* attribute of both forms set to False.

```
//Display second form in a literal
private void butExecute Click(object sender, System.EventArgs e)
```

```
{
  System.IO.StringWriter swrTarget = new System.IO.StringWriter();
  // Execute a Web form, store the results (optiona 2nd parameter).
  Server.Execute("Table.aspx", swrTarget);
  // Display the result in a literal control.
  litTarget.Text = "<h2>Table Results</h2>" + swrTarget.ToString();
 }
```

4.11.7.9. Displaying a Page in a New Browser Windows.

4.11.7.9.1. Use the *onclick="window.open()"* client-side method. The call has the following format

```
<INPUT style="Z-INDEX: 102; LEFT: 55px; WIDTH: 81px; POSITION: absolute; "
onclick="window.open('webform2.aspx')" type="submit"
value="New Window">
```

4.11.7.9.2. When utilized in a WebForm you can use variable as the target window:

```
<INPUT style="Z-INDEX: 102; LEFT: 55px; WIDTH: 81px; POSITION: absolute; TOP: 156px;
HEIGHT:
24px" onclick="window.open('<%# urlTarget %>')" type="submit"
value="New Window">
```

(Check how to use class-like code to set attributes of the new window in pg. 182)

5. Storing and Retrieving data with ADO.NET

5.1. Use unTyped datasets only when the data source is supplied at run time rather than design time.

5.2. After defining a dataset as the source of a Datagrid, to fill data use:

```
private void Page_Load(object sender, System.EventArgs e)
   {
         // Fill the data set.
         sqlDataAdapter1.Fill(DataSet1);
         // Update the DataGrid.
         DataGrid1.DataBind();
   }
```

5.3. Because you are filling the data set and binding the data to the *DataGrid* control in the *Page_Load* event, you don' t need to maintain state information for the *DataGrid* control. Turning off state information improves performance because the data on the *DataGrid* does not have to be saved to the page' s *ViewState* between requests. Set the DataGrid' s *EnableViewState* property to False to turn off state maintenance for the control.

5.4. How to add a row in a Typed dataset.

```
// Using connection, adapter, and data set created in Design mode.
private void butAddRow_Click(object sender, System.EventArgs e)
{
    // Create a new row object for the Contacts table.
    DataSet1.ContactsRow rowNew = (DataSet1.ContactsRow)DataSet1.Contacts.NewRow();
    // Add data to the columns in the row.
    rowNew.ContactID = 42;
    rowNew.FirstName = "Danielle";
    rowNew.LastName = "Tiedt";
    rowNew.WorkPhone = "(111) 555-1212";
    DataSet1.Contacts.Rows.Add(rowNew);
}
```

5.5. Using the *FindBy* method to find and change (or delete) a row.

```
// Using connection, adapter, and data set created in Design mode.
   private void butChangeRow_Click(object sender, System.EventArgs e)
   {
         // Declare a row object.
         DataSet1.ContactsRow rowChange;
         // Get the row to change using the primary key.
         rowChange = DataSet1.Contacts.FindByContactID(42);
         // Change a field in the row.
         rowChange.WorkPhone = "(111) 555-9000";
        //To delete it use
        rowChange.Delete();
   }
```

5.6. Update the database in the Page_PreRender event to make sure all the control events in the page have been processed.

```
// Using connection, adapter, and data set created in Design mode.
   private void Page_PreRender(object sender, System.EventArgs e)
   {
         // Update the database with changes from the data set.
         sqlDataAdapter1.Update(DataSet1);
   }
```

5.7. Binding a DropDownList in Studio, code generated:

```
<asp:DropDownList id="drpContacts" runat="server" Width="384px" Height="22px"
  DataSource="<%# dsContacts %>" DataTextField='LastName' DataValueField="ContactID">
</asp:DropDownList>
```

5.8. To customize the values to display in a DropDownlList do the following.

```
private void Page_Load(object sender, System.EventArgs e)
{
 // Run first time page is displayed.
 if (!IsPostBack)
    {
    // Fill the Contacts data set.
    adptContacts.Fill(dsContacts);
    // For each row in the table...
    foreach (dsContacts.ContactsRow drowItem in dsContacts.Contacts)
     {
       // Create a new list item.
       ListItem lstNew = new ListItem();
       lstNew.Text = drowItem.FirstName + " " +  drowItem.LastName;
       lstNew.Value = drowItem.ContactID.ToString();
       drpContacts.Items.Add(lstNew);                // Add the list item to the drop-down list.
     }
    }
}
```

5.9. Isolation Level Settings in the transaction object

ReadUncommitted	Does not lock the records being read. This means that an uncommitted change can be read and then rolled back by another client, resulting in a local copy of a record that is not consistent with what is stored in the database. This is called a dirty read because the data is inconsistent.
Chaos	Behaves the same way as ReadUncommitted, but checks the isolation level of other pending transactions during a write operation so that transactions with more restrictive isolation levels are not overwritten.
ReadCommitted	Locks the records being read and immediately frees the lock as soon as the records are read. This prevents any changes from being read before they are committed, but it does not prevent records from being added, deleted, or changed by other clients during the transaction. This is the default isolation level.
ReapeatableRead	Locks the records being read and keeps the lock until the transaction completes. This ensures that the data being read does not change during the transaction.
Serializable	Locks the entire data set being read and keeps the lock until the transaction completes. This ensures that the data and its order within the database do not change during the transaction.

5.10. Saving points in transactions

```
  //To set a save point within a SQL transaction, use the Save method:

transDelete.Save("FirstStep");

  // To restore a SQL transaction to a save point,
  // specify the name of the save point in the Rollback method:

transDelete.Rollback("FirstStep");
```

5.11. Enterprise Transactions

Because transactions can span multiple Web forms, or even multiple components within a distributed application, ASP.NET provides a way for Web forms to work with the DTC.

5.12. A way to deal with error handeling when updating a database

```
  try
    {
        // Modify the database.
      adptContacts.Update(dsContacts);
    }
       // Occurs if ContactID is not unique.
  catch (ConstraintException )
    {
      litStatus.Text = "ContactID is not unique"
    }
      // The database may be locked by another session, so let the user try again.
  catch (DBConcurrencyException )
    {
      litStatus.Text = "The database is currently locked. Wait a few seconds and then click Add
again.";
    }
      // General data exception, so report it.
  catch(DataException ex)
    {
      litStatus.Text = "The following database error occurred:<br>" + ex.Message + "<br>" +
```

```
"Correct the error and click Add to add the contact " +
                "or click Cancel to abort.<br>";
     Trace.Warn("Error", "Data exception", ex);
    }
```

6. Catching and correcting errors

6.1. To find what exceptions to use from the Debug menu, choose Exceptions. Visual Studio .NET displays the Exceptions dialog box

6.2. Application Exceptions

6.2.1. Using an ApplicationException

```
throw new ApplicationException("User is already logged on.");
.
.
catch (ApplicationException ex)
```

6.2.2. Extending the ApplicationException

The new exception class provides only its own constructor to set the default message to display. This is a standard practice.

```
public class UserLoggedOnException : System.ApplicationException
{
   // Exception constructor (overloaded).
  public UserLoggedOnException() :   this("The user is already logged on to the server",
null)
  {
  }

  public UserLoggedOnException(string message) : this(message, null)
  {
  }

  public UserLoggedOnException(string message, Exception inner) :   base(message, inner)
  {
  }
}
```

6.3. Using Error Events

6.3.1. There are the following error events

Page_Error	An unhandled exception occurs on the page. This event procedure resides in the Web form.
Global_Error	An unhandled exception occurs in the application. This event procedure resides in the Global.asax file.
Application_Error	An unhandled exception occurs in the application. This event procedure resides in the Global.asax file.

6.3.2. Use the following methods of the Server object to get information about the error.

GetLastError()	Get the last exception that occurred on the server.
ClearError()	Clear the last exception that occurred on the server. Invoking ClearError handles the exception so it does not trigger subsequent error events or appear to the user in the browser.

```
private void Page_Error(object sender, System.EventArgs e)
{
      // Get the error.
    Exception ex = Server.GetLastError();
      // Do something, here the message is stored to be displayed in the redirected page.
    Session["Error"] = ex.Message;
      // Clear the error.
    Server.ClearError();
      // Redisplay this page.
    Response.Redirect("ErrorEvent.aspx");
}

//Redirected Page: ErrorEvent.Aspx

private void Page_Load(object sender, System.EventArgs e)
{
      // Display error. if any.
    if(Session["Error"] != null)
    {
        litError.Text = "<p>The following error occurred:</p>" + Session["Error"].ToString
```

```
();
          // Clear the Session state variable.
        Session["Error"] = null;
    }
}
```

6.4. Using Error Pages

Error pages handle exceptions that occur outside of the Web application because of Internet-related problems

6.4.1. Application-wide error pages

6.4.1.1. Using IIS and setting the "Custom Error" tab of the properties windows for the Web application.

6.4.1.2. Defining the pages to display in the <CustomErrors> tag of the Weg.config file:

```
<customErrors mode="On" defaultRedirect="ErrDefault.aspx">
        <error statusCode="401" redirect="ErrUnauthorized.aspx" />
        <error statusCode="404" redirect="ErrPageNotFound.aspx" />
        <error statusCode="500" redirect="ErrServer.htm" />
</customErrors>
```

6.4.2. Page-level error pages

Utilized to display a specific page when an unhanded exception occurs on a Web form. Use the Server object' s *GetLastError* and *ClearError* methods from the error page to get the error.

-Set the errorPage attribute for a Web form

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbExceptionSnippet.WebForm1" errorPage="errDefault.aspx"%>
```

```
// **** Webform1.aspx
private void butError_Click(object sender, System.EventArgs e)
{
   // Cause exception.
 throw new System.Net.WebException();
}

// **** ErrDefault.aspx
private void Page_Load(object sender, System.EventArgs e)
{
    // Display the error that occurred.
 litError.Text = "<p>The following error occurred:</p>" + Server.GetLastError().Message;
   // Clear the error.
 Server.ClearError();
}
```

6.5. Logging Exceptions

Creates an exception log, which is a list of handled exceptions.

6.5.1. Turning tracing On for an entire application.

Make the following changes to the <trace> tag in the **web.config** file.

```
<trace enabled="true" requestLimit="20" pageOutput="false" traceMode="SortByTime"
localOnly="true" />
```

- enabled="true" – Turns on tracing
- pageOutput="false" – Send output to the **Trace.axd** file instead of to the page.
- RequesLimit="20" – Trace only for the first 20 requests
- LocalOnly="true" – The log file can see only in the local server.

6.5.2. Turning tracing On for an specific WebForm

Set the *enabled* and *pageOutput* web.config attributes as showed, and set the **Page** *trace* = true.

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
Inherits="csTracing.WebForm1" trace="True"%>
```

- trace=false – Sends the output to a log instead of to the page

6.5.3. Writing messages to the Trace log.

Use the *Write* and *Warn* methods. The first one displays messages in black and the other in red.

The *Page.Trace* property returns a *System.Web.TraceContext* object

```
   //Manage errors using the Warn method
private void Page_Error(object sender, System.EventArgs e)
{
      // Write a message to the trace log.
    Trace.Warn("Error", "", Server.GetLastError());
      // Clear the error so the application can continue.
    Server.ClearError();
      // Redisplay the page.
    Response.Redirect("Trace.aspx");
}

  //Create an exception to test the former code
private void butError_Click(object sender, System.EventArgs e)
```

```
{
    throw new System.IO.FileNotFoundException();
}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  //Testing if Trace is enabled before getting additional information
private void Page_Error(object sender, System.EventArgs e)
{
    if (Trace.IsEnabled)
    {
        string strMessage;
        if (Request.Browser.AOL)
            strMessage = "AOL Browser";
        else
            strMessage = "Non-AOL Browser";
        // Write a message to the trace log.
        Trace.Write("Error", strMessage, Server.GetLastError());
    }
    // Clear the error so the application can continue.
    Server.ClearError();
    // Redisplay the page.
    Response.Redirect("Trace.aspx");
}
```

7. Advanced Web Forms Programming.

7.1. Steps to create and update cookies. (System.Web.HttCookie)

```
private void Page_Load(object sender, System.EventArgs e)
{
      // (1) Check if browser accepts cookies.
      if (Request.Browser.Cookies)
       {
        // (2) If the cookie does not exist...
        if (Request.Cookies["LastVisit"] == null)
           {
           // (3) Create cookie.
          HttpCookie cookLastVisit = new HttpCookie("LastVisit", DateTime.Now.ToString());
          // (4) Set the expiration to tommorrow.
          cookLastVisit.Expires = DateTime.Now.AddDays(1);
          // (5) Add to cookies collection.
          Response.Cookies.Add(cookLastVisit);
          // Display message.
          Response.Write("This is your first visit.");
          }
        else
       {
          // Get the cookie.
          HttpCookie cookLastVisit = Request.Cookies["LastVisit1"];
          // Display a message showing time of last visit.
          Response.Write("You last visited this page: " + cookLastVisit.Value);
          // Update the cookie on the client.
          Response.Cookies["LastVisit"].Value = DateTime.Now.ToString();
          Response.Cookies["LastVisit"].Expires = DateTime.Now.AddDays(1);
       }
      }
     else
      {
        Response.Write("Your browser does not accept cookies.");
      }
}
```

7.1.1. Cookies are case sensitive and can hold up to 4096 bytes

7.1.2. By default Cookies expire when the session ends. Setting *Expires* to the *DateTime.MaxValue* means that the cookie never expires. You can remove the cookie from the client' s machine by resetting the *Expires* property to the current time.

```
// Set cookie to expire immediately.
      Response.Cookies["LastVisit"].Expires = DateTime.Now;
```

7.1.3. Using keys with cookies

```
//Setting Cookies with keys
private void butOK_Click(object sender, System.EventArgs e)
{
      // Create a cookie cookie.
      HttpCookie cookUserInfo = new HttpCookie("UserInfo");
      // Fill in the keys from the form data.
      cookUserInfo["FirstName"] = txtFirstName.Text;
      cookUserInfo["LastName"] = txtLastName.Text;
      cookUserInfo["Street"] = txtStreet.Text;
      cookUserInfo["City"] = txtStreet.Text;
```

```
      cookUserInfo["State"] = drpState.SelectedItem.Value;
      cookUserInfo["ZIP"] = txtZIP.Text;
      // Set the expiration.
      cookUserInfo.Expires = DateTime.Now.AddDays(30);
      // Add the cookie.
      Response.Cookies.Add(cookUserInfo);
}

//Retrieving
private void butGetData_Click(object sender, System.EventArgs e)
{
      // Get the cookie.
      HttpCookie cookUserInfo  = Request.Cookies["UserInfo"];
      // Fill in the fields.
      txtFirstName.Text = cookUserInfo["FirstName"];
      txtLastName.Text = cookUserInfo["LastName"];
      txtStreet.Text = cookUserInfo["Street"];
      txtCity.Text = cookUserInfo["City"];
      drpState.SelectedItem.Value = cookUserInfo["State"];
      txtZIP.Text = cookUserInfo["ZIP"];
}
```

7.1.4. Creating unique keys to identify users

Use the *NewGuid* from the *System.Guid* name space to generate unique Ids

```
private void butOK_Click(object sender, System.EventArgs e)
{
      // Get the request cookie.
      HttpCookie cookUserID  = Request.Cookies["UserID"];
      // If it doesn't exist, create it.
      if (cookUserID == null)
            // Create a new cookie with a new GUID.
            cookUserID = new HttpCookie("UserID", System.Guid.NewGuid().ToString());
      // Set the expiration.
      cookUserID.Expires = DateTime.Now.AddDays(30);
      // Add the cookie to the response.
      Response.Cookies.Add(cookUserID);
      // Save the user info from the form data.
      SetUserInfo(cookUserID.Value);
}
```

7.2. Creating a file to store user information.

7.2.1. Create an XML file and schema using studio

7.2.2. Retrieve the XML file into a dataset

```
private DataSet GetUserData()
{
      // Set the path of the XML file and XML schema.
      string strPath  = Server.MapPath(Request.ApplicationPath);
      // Declare a data set.
      DataSet dsUsers = new DataSet();
      // Apply the XML schema to data set.
      dsUsers.ReadXmlSchema(strPath + "\\UserInfo.xsd");
      // Read the XML into the data set.
      dsUsers.ReadXml(strPath + "\\UserInfo.xml");
      return dsUsers;
}
```

7.2.3. Use the dataset to add new data (use the find method)

```
private void SetUserInfo(string UserID)
{
      // Set the path of the XML file and XML schema.
      string strPath  = Server.MapPath(Request.ApplicationPath);
      // Get the Users data set.
      DataSet dsUsers  = GetUserData();
      // Find the row in the data set.
      DataRow rowUser = dsUsers.Tables["User"].Rows.Find(UserID);
      // If the row is not found, then create a new row.
      if (rowUser == null)
      {
            rowUser = dsUsers.Tables["User"].NewRow();
            dsUsers.Tables["User"].Rows.Add(rowUser);
      }
      // Save data from form fields.
      rowUser["FirstName"] = txtFirstName.Text;
      rowUser["LastName"] = txtLastName.Text;
      rowUser["Street"] = txtStreet.Text;
      rowUser["City"] = txtCity.Text;
      rowUser["State"] = drpState.SelectedItem.Text;
      rowUser["ZIP"] = txtZIP.Text;
      rowUser["ID"] = UserID;
```

```
      // Write the XML from the data set.
      dsUsers.WriteXml(strPath + "\\UserInfo.xml");
}
```

7.2.4. Use the dataset to retrieve data

```
void GetUserInfo(string UserID)
{
      // Get the Users data set.
      DataSet dsUsers  = GetUserData();
      // Find the row in the data set.
      DataRow rowUser  = dsUsers.Tables["User"].Rows.Find(UserID);
      // If user wasn't found, exit.
      if (rowUser == null) return;
      // Add data to form fields.
      txtFirstName.Text = rowUser["FirstName"].ToString();
      txtLastName.Text = rowUser["LastName"].ToString();
      txtStreet.Text = rowUser["Street"].ToString();
      txtCity.Text = rowUser["City"].ToString();
      drpState.SelectedItem.Text = rowUser["State"].ToString();
      txtZIP.Text = rowUser["ZIP"].ToString();
}
```

7.3. Sending Mail

7.3.1. Using the mailto protocol.

```
mailto:address[?key=setting][&key=setting] …
```

Parameters:

Key	Specifies
SUBJECT	The text to appear in the message' s subject line.
CC	A list of addresses to copy the message to. Multiple addresses are separated by semicolons (;).
BCC	A list of addresses to send blind copies to. Multiple addresses are separated by semicolons (;).
BODY	The text of the message.

```
<A href="mailto:someone@microsoft.com?SUBJECT=Sending from a client&BODY=Some message text.">

<asp:HyperLink ID="hypMail" NavigateUrl="mailto:someone@microsoft.com?
   SUBJECT=Mailing a Webform&amp;BODY=Some message text." Runat="server">
Send mail.
</asp:HyperLink>
```

7.3.2. Sending mail from the server

7.3.2.1. Using the static *Send* method of *SmtpMail* class from the *System.Web.Mail* name space. The e-mail addresses are not validated.

```
SmtpMail.Send("someone@microsoft.com", "jesse@contoso.com", "Subject line",
                                                              "Message text.");
```

7.3.2.2. Using the *MailMessage* class.

```
private void butSendMail_Click(object sender, System.EventArgs e)
{
  // Create the message.
  MailMessage mailNew = new MailMessage();
  // Set the message properties.
  mailNew.From = "someone@microsoft.com";
  mailNew.To = "jesse@contoso.com";
  mailNew.Subject = "This is the subject text.";
  mailNew.Body = "This is the message text.";
  // Create an attachment.
  MailAttachment atcItem =
          new MailAttachment   ( Server.MapPath(Request.ApplicationPath) +
"\\joey.jpg" );
  // Attach it to the message.
  mailNew.Attachments.Add(atcItem);
  // Send the message.
  SmtpMail.Send(mailNew);
}
```

7.4. Using Frames

7.4.1. Create frames using Studio

7.4.2. To validate if a browser supports frames use

```
private void butDisplayPage_Click(object sender, System.EventArgs e)
{
      // If the client's browser supports frames, display the frameset.
    if (Request.Browser.Frames)
          Server.Transfer("frames.htm");
      // Otherwise, display a plain HTML version.
    else
```

```
            Server.Transfer("noframes.htm");
}
```

Or using the HTML <noframes> element tag.

```
<frameset rows="64,*">
      <frame name="banner" src="Banner.aspx" scrolling="no" noresize>
      <frameset cols="150,*">
            <frame name="contents" src="Contents.aspx">
            <frame name="main" src="Body.aspx">
      </frameset>
      <noframes>
            <p>
            This HTML frameset displays multiple Web pages. To view this
            frameset, use a Web browser that supports HTML 4.0 and later.
            </p>
      </noframes>
</frameset>
```

7.4.3. Targeting Frames with Links.

To link between pages displayed in frames, add a `TARGET` attribute to the link. For example, the following HTML displays a list of links that target the main frame in a frameset:

```
<h3>Contents</h3>
<P><A href="Body1.aspx" target="main">Body1</A></P>
<P><A href="Body2.aspx" target="main">Body2</A></P>
<P><A href="Body3.aspx" target="main">Body3</A></P>
```

7.5. Using Client-side scripts

7.5.1. Testing the browser for script compatibility

```
private void Page_Load(object sender, System.EventArgs e)
{
      if (Request.Browser.VBScript)
            Response.Redirect("VBScripts.htm");
      else if (Request.Browser.JavaScript)
            Response.Redirect("JScripts.htm");
      else
            Response.Redirect("NoScripts.htm");
}
```

Because the test returns true even if scripting was disabled, using the following script will redirect to the right page.

```
<html>
   <script>
      window.navigate("scripts.aspx")
   </script>
   <!-- If scripting is enabled, the following is never displayed -->
   <head>
   <title>Scripts</title>
   </head>
   <body MS_POSITIONING="FlowLayout">
   <h2>Scripting is turned off.</h2>
   <p>Your Internet security options specify that your browser will not run
   scripts, therefore you cannot view the page you requested. To turn
   scripting on, reset your browser's Internet security options to Medium, or   enable active
scripting in the custom security settings.</p>
   <p><a href="CheckBrowser.htm">Click here</a> to try again.</p>
   </body>
</html>
```

Using Javascript from the Server to check if the Browser has scripting enabled.

```
Response.Write("<script>window.navigate('frameset.htm')</script>");
Response.Write("Scripting is disabled due to your browser security settings.<br>" +
               "Change your browser's security settings allow scripting.<br><br>");
```

7.5.2. When scripts refer to an element, such element must appear before the script refers to it.

7.6. Consuming XML web services

7.6.1. They are located by using a UDDI registry. (Universal Description Discovery and Integration.

7.6.2. Using XML services from Client-Side scripts.

8. Maintaining Security

8.1. Authenticating and Authorizing users

Authentication is the process of identifying users. Authorization is the process of granting access to those users

8.1.1. ASP.NET uses **Impersonation** to assign a user account to an unknown user. The default anonymous user by default is named **IUSER_*machinename***.

8.1.2. To restrict the access of anonymous users use the Windows file permissions.

8.1.3. There are three major ways to authenticate and authorize users in ASP.NET
- Windows Integrated Authentication. Using regular network security.
- Forms authentication. Using a logon Web form.
- Passport authentication. Using Microsoft passport service

8.1.4. Using Authentication with HTML files.
To authenticate users who access HTML pages you need to register those files (htm and html extensions) to the ASP.NET executable (aspnet_isapi.dll). You do it using IIS.

8.2. Using Windows Authentication

8.2.1. It is the default authentication method when creating a new Web application; here ASP.NET checks the project' s Web.config authorization list to see which network users are allowed to access the application.

8.2.2. When a user is authorized, ASP.NET issues an authorization certificate in the form of a cookie that persists for the duration of the user' s session. The user' s session ends when the browser closes or when the session times out.

8.2.3. Examples of using the `<authorization>` element in the Web.config file. The * character indicates all users, the ? character indicates unauthorized users.

```
<authentication mode="Windows" />

<authorization>
                <!-- Allow two users from the network domain. -->
<allow users="Deanna Meyer, Michael Emanuel" />
<deny users="*"  />                         <!-- Deny anyone else. -->
</authorization>
```

8.2.4. Using Role-based authorization over Windows defined roles.

```
<authentication mode="Windows" />

<authorization>
    <allow roles="Administrators" />    <!-- Allow Administrators. -->
    <deny users="*"  />             <!-- Deny anyone else. -->
</authorization>
```

8.2.5. Get information of authenticated and authorized users using the `User` object

```
User.Identity.IsAuthenticated.ToString();   //get True or False
User.Identity.Name;      //gets user name like OEMCOMPUTER\Jeyy Web
User.Identity.AuthenticationType    //Get a type like: Negotiate
   //To get the role do
if(User.IsInRole("Administrators"))
    // Do something
User.Identity.AuthenticationType    //Authentication method utilized.
```

8.2.6. IIS authorization settings are evaluated first than Web.config settings. The most restrictive settings will be utilized.

8.3. Using Forms Authentication

8.3.1. This authentication uses the name space *System.Web.Security*. Here you have to create a web form and a file or database to get and validate the users. Also the Web.config file has to be modified as follows:

```
<authentication mode="Forms" >              <!-- Set authentication mode -->
  <forms loginUrl="LogIn.aspx" >            <!-- Specify a log on form -->
       <credentials passwordFormat="Clear"> <!-- Create a user list -->
              <user name="June" password="JuneBug"/>
              <user name="Walter" password="Halifax"/>
       </credentials>
  </forms>
</authentication>

<authorization>
  <deny users="?" />  <!—Deny all unauthenticated users -->
</authorization>
```

8.3.2. Forms Authentication Settings in Web.config

Element	Attribute	Description
<authentication>	mode	Set to Forms to enable Forms authentication.
<forms>	name	The name of the cookie to store the user' s credential (default = .authaspx). Specify a unique name when several applications are using Forms authentication on the same server
	loginUrl	Login for to display (default = Default.aspx)
	protection	Use to set how ASP.NET protects the authentication cookie stored on the user' s machine. The default is All, which performs encryption and data validation. Other possible settings are Encryption, Validation, and None.
	timeout	Minutes the authentication cookie persists on the user' s machine. The default is 30, being renewed if it receives a request from the user and more than half of the allotted time has expired.
	path	Path to store the cookie on the user' s machine. default is "\".
<credentials>	passwordFormat	The algorithm to encrypt the password. The default is SHA1. Other possible settings are MD5 and Clear (which prevents encryption).
<users>	name	Use to set the name of the user.

	password	Use to set the password for the user.

8.3.2.1. Once a user is authorized, ASP.NET issues an authorization certificate in the form of a cookie that persists for an amount of time specified by the authentication settings in Web.config.

8.3.2.2. The code of a login Web form would be like this: (based in a list in Web.config)

```
// Include this line at the beginning of the module.
   using System.Web.Security;

private void butSignOn_Click(object sender, System.EventArgs e)
{
      // Authenticate username/password from <credentials>.
      if (FormsAuthentication.Authenticate(txtUserName.Text, txtPassword.Text))
            // If found, display the application's Start page.
          FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, true);
      else
      {
            // Otherwise, clear the password.
            txtPassword.Text = "";
            // If third try, display "Access Denied" page.
            if (System.Convert.ToInt32(ViewState["Tries"]) > 1)
                  //This HTML page must be outside the scope of the application
                Response.Redirect("Denied.htm");
            else
                  // Otherwise, increment number of tries.
                ViewState["Tries"] = System.Convert.ToInt32(ViewState["Tries"]) + 1;
      }
}


//To sign out or to remove the cookie in the client
private void butSignOut_Click(object sender, System.EventArgs e)
{
      // Remove authentication cookie.
   FormsAuthentication.SignOut();
      // Redirect back to this page (displays log in screen).
   Response.Redirect("UserInfo.aspx");
}
```

8.3.2.3. To save an encrypted password in a database use:

```
String myPsw = FormsAuthentication.HashPasswordForStoringInConfigFile (strPassword "SHA1");
```

Algoritms available are SHAI and MD5.

8.4. Using passport authentication.

8.4.1. Install the Passport SDK.

8.4.2. Set the Web.config to passport authentication mode

```
<authentication mode="Passport" />

<authorization>
    <deny users="?" /> <!-- Deny unauthenticated users -->
</authorization>
```

8.4.3. Use the `Global.asax`'s `PassportAuthentication_OnAuthenticate` event to access the user's data. The data is stored in five cookies in the client's machine.

```
// Add to the begging of module
   using System.Web.Security;

protected void PassportAuthentication_OnAuthenticate(Object sender,
                                                     PassportAuthenticationEventArgs e)
{
   // Get Session's passport identity if authenticated.
 if (e.Identity.IsAuthenticated)
   {
     Response.Write("Name: " + e.Identity["FirstName"] + e.Identity["LastName"] + "<br>");
     Response.Write("Address: " + e.Identity["City"] + + e.Identity["PostalCode"] + "<br>");
     Response.Write("Email: " + e.Identity["PreferredEmail"] + "<br>");
     Response.Write("Passport ID: " + e.Identity.Name + "<br>");
   }
}
```

8.4.4. Remove the Passport cookies in a SingOut form.

```
private void butSignOut_Click(object sender, System.EventArgs e)
{
   // Sign out by deleting Passport cookies.
   Response.Cookies["MSPProf"].Expires = DateTime.Now;
   Response.Cookies["MSPAuth"].Expires = DateTime.Now;
   Response.Cookies["MSPSecAuth"].Expires = DateTime.Now;
```

```
    Response.Cookies["MSPProfC"].Expires = DateTime.Now;
    Response.Cookies["MSPConsent"].Expires = DateTime.Now;
    // Redisplay this page (goes back to sign-in).
    Response.Redirect("UserInfo.aspx");
}
```

8.5. Providing Secure Communication

8.5.1. Generate a certificate request from IIS.
It has to be created at the server root before can be configured for subordinate sites (also those sites can have separate certificates).

8.5.2. Request a certificate from a Certificate Authority.

8.5.3. Install the certificate on the server using IIS.

8.5.4. Install the certificate on browsers if you are using a test certificate.

8.5.5. Use the secure protocol (https:) when accessing secure pages in your application. You can define a normal http page and then redirect to another with hhtps

9. Building an Deploying Web Applications

9.1. Steps to deploy an application

9.1.1. Identify the application by setting the assembly attributes of the **AssemblyInfo** file. This identifies the dll.

9.1.2. Set the **build option to "release"** using Studio in the Configuration Manager window from the Build menu

9.1.3. Web.config changes.
The application's Web.config file overrides the Machine.config, and other Web.config files from the IIS root or any parent application. Making changes to the Web.config file resets the application.

9.1.3.1. Change the *Compilation* element in the Web.config file

```
<compilation defaultLanguage="vb" debug="false" />
```

9.1.3.2. Change other attributes as necessary.

Web.config Attributes

compilation	Set the build type to debug or release
customErrors	Display custom error pages in response to HTTP response codes
authentication	Set the type of authentication to use to identify users
authorization	List user names or user roles that are authorized to run the application
trace	Enable tracing to help diagnose errors or tune performance
sessionState	Determine how ASP.NET stores Session state variables
globalization	Set cultural information for localized applications

9.2. Deploying a Web application

9.2.1. Use a registered hosting service from Studio

9.2.2. For own web servers

9.2.2.1. Use IIS to set up a virtual folder. Make sure the .NET framework in the server is of the same version of the one used to compile the application.

9.2.2.2. Copy the Web application to the virtual folder. Copy the assemblies (.dll) to the /bin folder, and make sure the application has a default.html or default.aspx file (Or define in IIS the default to be used).

9.2.2.3. Install shared components.

- For COM components. Copy and register them in the server. You can use the utility RegSvr32. (Example: `RegSvr32 myComp.dll`)
- For weakly-named .NET components. Copy them to the /bin directory.
- For strongly-named .NET components. Install then in the global assembly cache which is a special subfolder (named assembly) in the Windows folder- by dragging them using Windows Explorer- or by using the utility GacUtil.exe, (example: `GacUtil –i MyServerControls.dll`).

9.2.2.4. Assign security privileges.
Because the application runs using the ASPNET account, some resources maybe restricted for this account. There are 3 options to grant additional permissions

- Grant the ASPNET user access to the required files. To use this option, the server must be using the NTFS file system.
- Change the group the ASPNET user belongs to.
- Use impersonation to run the process as another user by changing the Web.config file.
  - Here the application run as the WebFlyr user account

```
<identity impersonate="true" name="localhost\WebFlyr" password="hpB14dQi" />
```

  - If the application uses Windows authentication, you can impersonate using the authenticated user. For example this setting tells the application to use the permissions of the user who signed on to the application.

```
<identity impersonate="true" />       <!-- Use impersonation. -->

<authorization>
      <deny users="?"  />                <!-- Require authentication. -->
</authorization>
```

9.2.3. Maintaining a Deployed application

9.2.3.1. Process recycling is the technique of shutting down and restarting an ASP.NET worker process (aspnet_wp.exe) that has become inactive or is consuming excessive resources. This is done by changing the ProcessModel element in the Machine.config file. Process Recycling Attributes.

timeout	The amount of time (hh:mm:ss) before the process is shut down and restarted
shutDownTimeOut	How much time each process has to shut itself down.
requestLimit	The number of queued requests to serve before the process is shut down and restarted
restartQueueLimit	The number of queued requests to retain while the process is shut down and restarted.
memoryLimit	The percentage of physical memory the ASP.NET process is allowed to consume before that process is shut down and a new process is started.
responseRestartDeadlocInterval	The amount of time to wait before restarting a process that was shut down because it was deadlocked.
responseDeadlockInterval	The amount of time to wait before restarting a process that is deadlocked.

9.2.3.2. To tune performance you can do:

9.2.3.2.1. Setting Cache options. Use the FrontPage Server Extensions to control how pages are cached in the server' s memory.

9.2.3.2.2. Changing Application protection level. Use the IIS Application Settings to control the application protection level. A higher level of protection make an application more reliable but slower.

9.2.3.2.3. Adjusting processModel element attributes in the Machine.config file. In general lowering any of the following attributes allows the server to handle less client requests more quickly.

requestQueueLimit	The number of queued requests allowed before ASP.NET returns response code 503 (Server too busy) to new requests
clientConnectedCheck	The amount of time (hh:mm:ss) to wait before checking if a client is still connected
maxWorkerThreads	The maximum number of threads per processor
maxIOThreads	The maximum number of I/O threads per processor

9.2.3.2.4. Adjusting `sessionState` element attributes in the Web.config file. Turn off Session state tracking if the application is not using it. To turn off Session state, in the application' s Web.config file, set the sessionState element' s `mode` attribute to “Off”, as shown here:

```
<sessionState
   mode="Off"
   stateConnectionString="tcpip=127.0.0.1:42424"
   sqlConnectionString="data source=127.0.0.1;user id=sa; password="
   cookieless="false"
   timeout="20"
/>
```

9.2.3.2.5. General tips

- Turn off debugging for deployed applications.
- Avoid round trips between the client and server.
- Turn off Session state if it is not needed.
- Turn off ViewState for server controls that do not need to retain their values.
- Use stored procedures with databases
- Use SqlDataReader rather than data sets for read-forward data retrieval.

9.2.4. Deploying Across multiple Servers

Scalability is the ability to add capacity to an application.

9.2.4.1. Web Garden. An application running in a server with several CPUs. To do it change the Machine.config file:

```
<processModel
      WebGarden=”true”           <enabled>
       CpuMask=”0xffffffff”      <CPU to use>
/>
```

9.2.4.2. Web Farm. An application running in multiple servers. To implement it is necessary to install load balancing

such as Network Load Balancing.

9.2.4.3. Sharing state information.

To share Application state save the data in a resource that is available to all the processes, such as an XML file or database. To share Session state you can use two techniques

9.2.4.3.1.1. Using a state server. You have to define the setting in the Web.config file and run the utility *aspnet_state.exe* in the Session state server

```
<sessionState
   mode="StateServer"
   stateConnectionString="tcpip=192.168.1.102:42"  ===>server location
   sqlConnectionString="data source=192.168.1.102;user id=sa;password="
   cookieless="false"
   timeout="20"
/>
```

9.2.4.3.1.2. Using a SQL database. You have to modify the Web.config file and run the *InstallSqlState.sql* utility to the Session state server. This utility installs the database that shares Session state information across processes.

```
<sessionState
  mode="SQLServer"
  stateConnectionString="tcpip=192.168.1.102:42"
  sqlConnectionString="data source=192.168.1.102;user id=sa;password="
  cookieless="false"
  timeout="20"
/>
```

10. Testing Web Applications

10.1. Types of Tests

Test type	Ensures that
Unit test	Each independent piece of code works correctly
Integration test	All units work together without errors
Regression test	Newly added features do not introduce errors to other features that are already working
Load test (also called stress test)	The product continues to work under extreme usage
Platform test	The product works on all of the target hardware and software platforms

10.2. Unit Testing.

Written by the developer who wrote the program unit (Properties, methods, events, and classes). With unit testing each piece of code is tested using the same programming language or a scripting language with a range of possible values reporting the errors at the unit level to facilitate fixing the problem.

10.3. Integration Testing.

10.3.1. Stubs are non-functional components that provide the class, property, or method definition used by the other component. Stubs are a kind of outline of the code you will create later and they are created to help other code work.

10.3.2. Drivers are simply test components that make sure two or more components work together. Later in the project, testing performed by the driver can be performed by the actual component.

10.3.3. A testing interface is a set of public properties and methods that you can use to control a component from an external testing program. Here you use *#if…#endif* directives to prevent to compile the code between into the final version.

10.4. Regression Testing

10.4.1. If a new component or a change to an existing component breaks one of the existing unit or integration tests, the error is called a regression.

10.4.2. The key to success with both integration and regression testing is to run the full set of tests frequently—if possible, as part of the nightly build. Detecting problems early and resolving them as they occur prevents one error from hiding others.

10.5. Load Testing

10.5.1. Created using Microsoft Application Center (ACT), and run a load of test on a Web Application.

10.6. Debugging

10.6.1. The *Debug* class and *Trace* classes of the *System.Diagnostics* namespace provide methods to display alerts and messages. By default, *Debug* methods and properties are automatically stripped out of code compiled for release. *Trace* methods and properties are retained in release code by default.

10.6.2. Use *Assert* to halt an application for an unexpected result. In a Web application, the alert is displayed on the server.

```
Debug.Assert(BoolenTest,"A Message here");
```

10.6.3. To record *Debug* and *Trace* messages on a deployed application, create a *TextWriterTraceListener* class and add it to the *Debug* or *Trace* classes' *Listeners* collection.

```
  //Direct Debug messages to the server' s console
Debug.Listeners.Add(New TextWriterTraceListener(Console.Out))
Debug.WriteLine("Starting tests.")
  //To write messages to a file
Debug.Listeners.Add(new TextWriterTraceListener("Results.log"));
Debug.WriteLine("Starting tests.");
Debug.Flush();
```

1.1.1. *Debug* and *Trace* have 6 static methods to write output:

```
  // Writes text to the Listeners collection.
Trace.Write("Trace Message 1");
  // Writes text and a carriage return to the Listeners collection.
Trace.WriteLine("Trace Message 2");
  // Writes text if the supplied expression is true
Debug.WriteIf(X==Y, "X equals Y");
  // Writes text and a carriage return if the supplied expression is true
Debug.WriteLineIf(X==Y, "X equals Y");
  // Writes output and displays a message box if the condition is false
Trace.Assert(X==Y, "X does not equal Y!");
  // Writes output and displays a message box unconditionally
Debug.Fail("Drive B is no longer valid.");
```

1.1.2. Trace messages can be indented by using the *Indent* and *UnIndent* methods

```
// Increases the IndentLevel by one
Trace.Indent();
```

1.1.3. Output from Trace and Debug statements is received by members of the *Trace.Listeners collection*, a collection of objects that are able to receive Trace output and process it. There are three kinds of Trace Listeners:

1.1.3.1. *DefaultTraceListener* . Used by Visual Studio.

1.1.3.2. *TextWriterTraceListener*. writes its output as text, either to a Stream object or to a TextWriter object.

```
  // This line opens the specified file or creates it if it does not exist.
System.IO.FileStream myLog = new System.IO.FileStream("C:\\myFile.txt",
                                        System.IO.FileMode.OpenOrCreate);
  // Creates the new TraceListener that specifies myLog as the target for output
TextWriterTraceListener myListener = new TextWriterTraceListener(myLog);
  // Adds myListener to the Listeners collection
Trace.Listeners.Add(myListener);
```

```
  //You have to flush after making a write
Trace.Flush
  //Or you can set autoflush to do it after every write
Trace.AutoFlush = true;
```

1.1.3.3. *EventLogTraceListener* . Sends the output to an event log

```
//Declare an instance of an EventLog object and assign it either an
//existing event log or a new event log.
  EventLog myLog = new EventLog("Debug Log"); //New log created
//Set the Source property for the EventLog.
//If the Source property is not set, an error will result.
  myLog.Source = "Trace Output";
//Create a new instance of EventLogTraceWriter that specifies the
//new event log as the target for Trace output.
  EventLogTraceListener myListener = new EventLogTraceListener(myLog);
//If necessary, set the Trace.AutoFlush property to true,
//or call Trace.Flush after each write.
```

10.6.4. Remote Debugging

To debug an application on a remote server, follow these steps:

- Install the Visual Studio .NET remote debugging server components on the Web application server.
- Configure the server permissions to allow you to debug applications remotely.
- Attach to the remote Web application process from your debug machine.

10.6.5. Use DevEnd.exe tu run Vstudio from command line:

```
DevEnd myProgram.sln /build Debug > logs\builders.log
```

11. Creating Custom Web Controls

11.1. Creating Web User Controls

Web user controls combine one or more server or HTML controls on a Web User Control page, which can, in turn, be used on a Web form as a single control.

11.1.1. Limitations

- A copy of the control must exist in each Web application project where it is used.
- User controls cannot be loaded in the Visual Studio .NET Toolbox; instead, you must create them by dragging the control from the Solution Explorer to the Web form.
- User control code is initialized after the Web form loads, which means user-control property values are not updated (control's *Page_Load* event) until after the Web form' s Page_Load event.

11.1.2. To use a User Control in the HTML portion of a Web form:
Register the User Control

```
<%@ Register TagPrefix="ucx" TagName="Spinx" Src="Spin.ascx" %>
```

Place an instance if it where is required

```
<ucx:Spin id="Spinx" runat="server" Value="5"></uc1:Spin>
```

11.1.3. User controls support flow layout only, so if you want to position controls on the page using grid layout, add an HTML Grid Layout Panel control to the user control as a container for the controls you want to position. Then by using "delegation" set its properties.

```
//*****User Control: HTML generated
<DIV id="pnlGrid" runat="server" style="WIDTH: 20px; POSITION: relative;
HEIGHT: 48px" ms_positioning="GridLayout">
   <asp:Button id="butUp" Text="^" runat="server" />
   <asp:Button id="butDown" Text="v" runat="server" />
</DIV>

//*****User Control: Code
//Define two properties to access the Panel's Style attribute
//(Style defines position)
public string StyleToSet
{
      get
      {
            // Return the containing Panel control's style attribute.
            return pnlGrid.Attributes["style"];
      }
      set
      {
            // Set the containing Panel control's style attribute.
            pnlGrid.Attributes["style"] = value;
      }
}

//**** Web Form: Using the user control
<UserControl:Spin id="Spin1" runat="server" Value="5"
  styleToSet="Z-INDEX: 101; LEFT: 248px; POSITION: absolute; TOP: 88px">
  </UserControl:Spin>
```

11.1.4. Create *Public* properties and events for the user controls to be used from the Web form. Store information in the *ViewState* to make it available between page displays, and because those values are loaded after the *Page*'s *Page_Load* event, use the *Page_PreRender* event to set and get User Control's *ViewState* values.

11.1.5. Adding events to the User Control.

```
//**** In the User control ****
public abstract class Spin : System.Web.UI.UserControl
      {
// Declare the event.
public event EventHandler Click;

// Method to raise event. Put only for convenience
// because it can be overriden if necessary by the form
protected virtual void OnClick(EventArgs e)
{
      if (Click != null)
      {
            Click(this, e);
      }
}

private void butDown_Click(object sender, System.EventArgs e)
{
      // Decrement the Value and Call the OnClick method.
      this.Value -= 1;
      OnClick(e);
}

private void butUp_Click(object sender, System.EventArgs e)
{
      // Increment the Value and call the OnClick method.
```

```
      this.Value += 1;
     OnClick(e);
}


//**** Code in the form ****
Protected Spin1 As Spin
     .
     .
Spin1.Click += Spin1_Click
     .
     .
private void Spin1_Click(object sender, System.EventArgs e)
{
      // Display the Spin control's value.
      TextBox1.Text = Spin1.Value.ToString();
}
```

11.2. Creating Composite Custom Controls

11.2.1. To create a Custom Control choose the VStudio template "Web Control Library" to create the following template. The template includes a class named *WebCustomControl1* that contains one property, which is named *Text*.

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

namespace csCompositeSnippet
{
   /// <summary>
   /// Summary description for WebCustomControl1.
   /// </summary>
[DefaultProperty("Text"),
  ToolboxData("<{0}:WebCustomControl1
   runat=server></{0}:WebCustomControl1>")]
   public class WebCustomControl1 : System.Web.UI.WebControls.WebControl
   {
         private string text;

          [Bindable(true),
               Category("Appearance"),
               DefaultValue("")]
         public string Text
         {
               get
               {
                     return text;
               }

               set
               {
                     text = value;
               }
         }

         // Render this control to the output parameter specified.
         // param name="output" -> The HTML writer to write out to
        protected override void Render(HtmlTextWriter output)
         {
               output.Write(Text);
         }
   }
}
```

Determines design-time settings.

Displays the custom control

11.2.2. To test the control create an ASP.NET project and add a project reference to the custom control project (which makes a copy of the custom control assembly to the /bin directory of the Web App).

11.2.3. Now add an instance of the control to the Web form:

Add a register directive

```
<%@ Register TagPrefix="Custom" Namespace="vbCompositeSnippet"
 Assembly="vbCompositeSnippet" %>
```

TagPrefix –	Just a group identificator, like "asp" is for ASP.NET server controls
NameSpace –	Project name and namespace within the custom control assembly that contains the controls to register
Assembly –	Name of the assembly (.dll) containing the customs controls.

Create an instance of the custom control on the Web form.

Use the TagPrefix (Custom) plus the class name (WebCustomControl1).

```
<Custom:WebCustomControl1 id="custTest" runat="server" />
```

11.2.4. Create the composite control appearance

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

namespace csCompositeSnippet
{
  [DefaultEvent("Click")]
        public class MathBox : System.Web.UI.WebControls.WebControl
        {
                TextBox txtMath = new TextBox();
                Button butSum = new Button();
                Label lblResult = new Label();

                // Declare the event.
                public event EventHandler Click;

                protected override void CreateChildControls()
                {
                // Add the sub controls to this composite control.
                     // Set the TextMode property and add textbox.
                     txtMath.TextMode = TextBoxMode.MultiLine;
                     Controls.Add(txtMath);
                     // Start a new line
                     Controls.Add(new LiteralControl("<br>"));
                     // Set the Text property and add the Button control.
                     butSum.Text = "Sum";
                     Controls.Add(butSum);
                     // Add event handler.
                     butSum.Click += new EventHandler(butSumClicked);
                     // Add Label and Literals to display result.
                     Controls.Add(new LiteralControl(" &nbspResult: <b>"));
                     Controls.Add(lblResult);
                     Controls.Add(new LiteralControl("</b>"));
                }

                //Always must be overridden, even if only calls the base.
                protected override void Render(HtmlTextWriter output)
                {
             EnsureChildControls();
             // Resize text box to match control width.
             txtMath.Width = this.Width;
             // Resize text box to match control height.
                   double dHeight = this.Height.Value - butSum.Height.Value;
             txtMath.Height = Unit.Parse(dHeight.ToString());
             // Render the control.
                   base.Render(output);
                }

                void butSumClicked(object sender, EventArgs e)
                {
                        // Call the Sum method.
                        Sum();
                        // Call the event method.
                        OnClick(EventArgs.Empty);
                }

                //"Bridge" method to call the event, a convenience in case
        // the call to the event needs to be overridden
                protected virtual void OnClick(EventArgs e)
                {
                        // Raise the event.
                        Click(this, e);
                }

                // MathBox properties and methods.

                [DefaultValue("0")]
                public string Text
                {
                        get
                        {
                                // Make sure child controls exist.
                                EnsureChildControls();
                                // Return the text in the TextBox control.
                                return txtMath.Text;
                        }
```

```
                        set
                        {
                                // Make sure child controls exist.
                                EnsureChildControls();
                                // Set the text in the TextBox control.
                                txtMath.Text = value;
                        }
                }

                char[] strSep = {'\r'};

                public string[] Values
                {
                        get
                        {
                                EnsureChildControls();
                                // Return an array of strings from the TextBox.
                                return txtMath.Text.Split(strSep);
                        }
                        set
                        {
                                EnsureChildControls();
                                // Set the text in the TextBox from an array.
                                txtMath.Text = String.Join(" ", value);
                        }
                }

                public string Result
                {
                        get
                        {
                                EnsureChildControls();
                                // Return the result from the Label.
                                return lblResult.Text;
                        }
                }

                public void Sum()
                {
                        EnsureChildControls();
                        // If there is text in the TextBox.
                        if (txtMath.Text.Length != 0)
                        {
                                // Break the text into an array, line by line.
                                string[] arrNums;
                                arrNums = txtMath.Text.Split(strSep);
                                double dblSum = 0;
                                // Add each element in the array together.
                                foreach (string strCount in arrNums)
                                {
                                // Use error handling to ignore non-number entries.
                                        try
                                        {
                                                dblSum += Convert.ToDouble(strCount);
                                        }
                                        catch
                                        {
                                        }
                                }
                                // Display the result in the label.
                                lblResult.Text = dblSum.ToString();
                        }
                        else
                                lblResult.Text = "0";
                }
        }
}
```

To force display in the Vstudio design windows, set an initial value

```
<Custom:MathBox id="mthTest" runat="server" Text="0" />
```

To use the MathBox' s event from a Web form, double-click the control in the test Web form and use the following event procedure:

```
private void mthTest_Click(object sender, System.EventArgs e)
{
      Response.Write(mthTest.Result.ToString());
}
```

11.2.5. Superclassing Controls

11.2.5.1. They are derived from a single control.

```
//Defining the control
public class SuperText : System.Web.UI.WebControls.TextBox
{

     public void Sort()
     {
         // Create an array.
        string[] arrText;
        // Put the words in the text box into the array.
        char[] strSep = {' '};
        arrText = this.Text.Split(strSep);
        // Sort the array.
        Array.Sort(arrText);
        // Join the string and put it back in the text box.
        this.Text = String.Join(" ", arrText);
     }
}

//After registering the control's assembly, instantiate it using

<Custom:SuperText id="superTest" runat="server"  />

<asp:Button ID="butSort" Runat="server" Text="Sort"  />

//To use the control in a button do
private void butSort_Click(object sender, System.EventArgs e)
{
       superTest.Sort();
}
```

11.3. Creating Rendered Custom controls

11.3.1. First create a solution containing a custom control project using the "Web Control Library" template.

11.3.2. Add an ASP.NET project

11.3.3. Add a project reference and then an HTML *Register* directive and control element to use the Custom control.

11.3.4. Create a custom control's visual interface, by overriding the base class's *Render* method.

11.3.4.1. HtmlTextWriter methods utilized to add HTML .

AddAttribute	Adds an HTML attribute to the next HTML element to render.
RenderBeginTag	Renders the begin tag of an HTML element for later writing by the WriteLine method.
RenderEndTag	Renders the end tag of an HTML element and writes the element and any rendered attributes that are pending. All rendered attributes are cleared after RenderEndTag.
WriteAttribute	Immediately writes an HTML attribute.
WriteBeginTag	Immediately writes the begin tag of an HTML element.
WriteEndTag	Immediately writes the end tag of an HTML element.
WriteFullBeginTag	Immediately writes the begin tag along with the closing bracket (>) for the HTML element.
WriteLine	Immediately writes a line of content. This is equivalent to the Write method, but WriteLine adds a newline character as well.
Write	Immediately writes a string.

11.3.4.2. First Approach: add HTML directly:

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

namespace csRenderedSnippet
{
   public class AlertButton : System.Web.UI.WebControls.WebControl
   {
      protected override void Render(HtmlTextWriter output)
      {
          // Write a title
          output.Write("<h3>Rendered Control</h3>");
          // Opens an Input HTML tag (inserts "<INPUT").
          output.WriteBeginTag("INPUT");
          // Write some attributes.
          output.WriteAttribute("value", "Custom Button");
          output.WriteAttribute("type", "button");
          output.WriteAttribute("onclick", "javascript:alert('Howdy!')");
          // Close the Input HTML tag (inserts ">").
          output.WriteEndTag("INPUT");
      }
   }
}
```

11.3.4.3. Second Approach: Rendering tag and attributes and then write them as a unit.

```
protected override void Render(HtmlTextWriter output)
```

```
{
   // Write a title
   output.Write("<h3>Rendered Control</h3>");
   // Add some attributes.
   output.AddAttribute("value", "Custom Button");
   output.AddAttribute("type", "button");
   output.AddAttribute("onclick", "javascript:alert('Howdy!')");
   // Opens an Input HTML tag (inserts "<INPUT").
   output.RenderBeginTag("INPUT");
   // Close the Input HTML tag (inserts ">").
   output.RenderEndTag();
}
```

11.3.5. Write code to store property values, respond to user actions, and get data from the user as needed

11.3.5.1. Storing Property Settings

Use the *ViewState* to store any property settings between page displays.

```
public string Text
{
      get
      {
      // If the property has been set
       if (ViewState["Text"] != null)
            // Return the settng.
            return ViewState["Text"].ToString();
       else
            // Otherwise return "".
            return null;
      }

      set
      {
            // Store the property setting.
            ViewState["Text"] = value;
      }
}
```

11.3.5.2. To retrieve contained text such as <b> **bold me** <b> do the following:

```
(1) add a ParseChildren attribute
(2) implement the InamingContainer interface
[ParseChildren(false)]
public class Red : System.Web.UI.WebControls.WebControl, INamingContainer
   {
   // the interface makes it possible to get the Controls array.

       protected override void Render(HtmlTextWriter output)
       {
          // Contained text is returned as a Literal control.
         LiteralControl litText ;

(3) retrieve the contained text using the Controls collection
         litText = (LiteralControl)Controls[0];
          // Make it red using the <font> element.
         output.Write("<font color='red'>" + litText.Text + "</font>");
       }
   }
```

To use the control in a Web form do:

```
<custom:red id="Red1" runat="server">Some red text</custom:red>
```

This example retrieves anything, including other controls using the Control collection

```
//Put contained controls inside a panel, center them on red background

[ParseChildren(false)]
   public class Center : System.Web.UI.WebControls.WebControl, INamingContainer
   {
     protected override void Render(HtmlTextWriter output)
         {
                  // Add some attributes for the panel.
               output.AddAttribute("align", "center");
               output.AddStyleAttribute("BACKGROUND-COLOR", "red");
                   // Start a panel
               output.RenderBeginTag("div");
                    // For each contained control.
               foreach (Control ctrItem in Controls)
               {
                // Render the control, this function outputs server
                      // control content to a provided HtmlTextWriter object
                     ctrItem.RenderControl(output);
               }
```

```
                output.RenderEndTag();
            }
    }
```

11.3.5.3. Responding to user actions

11.3.5.3.1. Control's cached events are raised after the *Page_Load* and post-back events have been handled. In this example *Change* is the default event and VStudio will generate the event procedure if you double-click the custom control in the Web form Designer.

```
//Custom Control class that defines an On<i>Change</i> event that is raised when the <i>Text</i> property
//changes
[DefaultEvent("Change")]
   public class RenderText : System.Web.UI.WebControls.WebControl
   {
(1) Declare and event to raise
         public event EventHandler Change;

         public string Text
         {
               get
               {
                // If the property has been set
                if (ViewState["Text"] != null)
                     // Return the setting.
                     return ViewState["Text"].ToString();
                else
                    // Otherwise return "".
                    return null;
               }
               set
               {
                     // Store the property setting.
                     ViewState["Text"] = value;
                     // Call event method.
                     OnChange(EventArgs.Empty);
               }
         }

         protected virtual void OnChange(EventArgs e)
         {
               if (Change != null)
(2) Raise the event.
                     Change(this, e);
         }

         protected override void Render(HtmlTextWriter output)
         {
            // Set the Input control's attributes.
            output.AddAttribute("value", this.Text);
            // Create an Input control element using above.
            output.RenderBeginTag("Input");
            // Close the Input control element (inserts />).
            output.RenderEndTag();
         }
   }
```

The following code handles the *Change* event from the Web form

```
private void txtTest_Change(object sender, System.EventArgs e)
         {
               Response.Write("Text changed!");
         }
```

11.3.5.3.2. Raising Post-Back events

They are raised after the Page_Load event.

```
(1) Implement the IPostBackEventHandler interface
[DefaultEvent("Click")]
public class AlertButtonThree : System.Web.UI.WebControls.WebControl,
                                IPostBackEventHandler
   {
(2) Declare a postback event to raise
         public event EventHandler Click;

(3) Render an HTML element that can detect client user events.
         protected override void Render(HtmlTextWriter output)
         {
           // Write a title
           output.Write("<h3>Rendered Control</h3>");
           // Add some attributes.
           output.AddAttribute("value", "Custom Button");
```

```
            output.AddAttribute("type", "button");
(4) Add attribute to raise Postback event on client.
            output.AddAttribute("onclick",
                              "javascript:alert('Howdy!');" +
                               Page.GetPostBackEventReference(this));
            // Opens an Input HTML tag (inserts "<INPUT").
            output.RenderBeginTag("INPUT");
            // Close the Input HTML tag (inserts ">").
            output.RenderEndTag();
          }
(5)  Raise the event from the RaisePostBackEvent method.
          // Part of the IPostBackEventHandler interface.
          public void RaisePostBackEvent(string eventArgument)
          {
                // Call the event method to raise event.
                OnClick(EventArgs.Empty);
          }

          protected virtual void OnClick(EventArgs e)
          {
                // Raise the event.
                if (Click != null)
                      Click(this, e);
          }

    }
```

The button's *onclick* event runs the following script

```
alert('Howdy!'); __doPostBack('winTest','');
```

The method *Page.GetPostBackEventReference(this)* generates the client's *__doPostBack()* method, which posts the page back to the server, where it is intercepted by the *RaisePostBackEvent* method, which in turn raises the *Click* event from within the control.
The following code in the Web form can handle the Click event.

```
private void altTest(object sender, System.EventArgs e)
          {
                Response.Write("Button clicked!");
          }
```

11.3.5.4. Getting data from the user

This example retains data entered by a user in a custom TextBox control rendered from an HTML *Input* element. Here by using a key in a collection you can identify the controls that are coming from other controls and check if its value has changed.

```
(1) implement the IPostBackDataHandler interface
  [DefaultProperty("Text"), DefaultEvent("Change")]
  public class RenderText : System.Web.UI.WebControls.WebControl,
                            IPostBackDataHandler
   {
         // Declare event.
         public event EventHandler Change;

          [DefaultValue("")]
         public string Text
         {
               get
               {
                // If the property has been set
                if (ViewState["Text"] != null)
                     // Return the settng.
                     return ViewState["Text"].ToString();
                else
                     // Otherwise return "".
                     return null;
               }

               set
               {
                     // Store the property setting.
                     ViewState["Text"] = value;
                     // Call event method. (Is this necessary?)
                     OnChange(EventArgs.Empty);
               }
         }

         protected override void Render(HtmlTextWriter output)
         {
            // Set the Input control's attributes.
```

```
            output.AddAttribute("value", this.Text);
(2) Add a name attribute to uniquely identify the HTML element to get
    data from.
            output.AddAttribute("name", this.UniqueID);
            // Create an Input control element using above.
            output.RenderBeginTag("Input");
            // Close the Input control element (inserts />).
            output.RenderEndTag();
        }

(3) Override the LoadPostBackData method and use it to get the data
    from the user.
        public bool LoadPostBackData(string postDataKey,
          System.Collections.Specialized.NameValueCollection
                                                    postCollection)
        {
           // If the user changes Input value,
           //update the text property.
              if (this.Text != postCollection[postDataKey])
              {
                    this.Text = postCollection[postDataKey];
               // Returning true invokes (4).
                    return true;
              }
           else
               // Returning False does not invoke (4).
               return false;
        }

(4) Override RaisePostDataChangedEvent method, even if you don't write
    code for it.
        public void RaisePostDataChangedEvent()
        {
(5) (optional) Raise an event to indicate that the data has changed.
           OnChange(EventArgs.Empty);
        }

        protected virtual void OnChange(EventArgs e)
        {
              if (Change != null)
                    // Raise event.
                    Change(this, e);
        }
```

11.4. Adding Custom controls to the ToolBox

11.4.1. Using Studio chose Toolbox from the Tools menu. The Customize ToolBox opens, there chose the ".NET framework components, locate the assembly (.dll) to add and click OK.

11.4.2. When you drag a control from the ToolBox, Studio generates the register directive in the Web form.

```
<%@ Register TagPrefix="cc1" Namespace="vbRenderedSnippet" Assembly="vbRenderedSnippet" %>
```

11.4.3. To assign an icon to the control, create a 16x16 bitmap file and Add the following code:

```
Using System.Drawing
    .
    .
//Attribute for the Control class
[ParseChildren(false), ToolboxBitmap("BitMapFileName")]
```

11.4.4. To change the TagPrefix used by the custom control in HTML add the following code to the project's AssemblyInfo file:

```
Using System.Web.UI
    .
    .
[assembly: TagPrefix("namespaceUtilized", "prefixToUse")]
```

12. Working with Multimedia

12.1. Playing Audio

12.1.1. Playing background sounds

Use the *bgsound* HTML tag but validate not to play it again each time the page is posted-back. Background sounds are additive, playing more that once are played simultaneously.
HTML script that plays the sound

```
<bgsound src="test.wav" id="bgTest" loop="infinite">
```

Validating the sound is played only the first time the page is posted

```
private void Page_Load(object sender, System.EventArgs e)
{
```

```
      // Plays theme first time page is displayed.
      if (!IsPostBack)
        Response.Write("<bgsound src='test.wav' id='bgsTest' loop='1'>");
}
```

12.1.2. Embedding Sounds

Use HTML tags to embed the default user's media player. You cannot control the playing of the sound, just hide it or set it to repeat.

```
<embed src="test.wav" id="wavTest" hidden="false" height="20" width="150"
autostart="false" type="audio/wav" loop="true"></embed>
```

12.1.3. Sounds as ActiveX Objects.

12.1.3.1. To install an ActiveX control (in an *Object* HTML tag), first add the ActiveX to the toolbox (using Customize ToolBox pop up menu), and drag it to the form.

```
<p>Here's the sound as a Windows Media Player object:</p>
<OBJECT id="objWMPlay" classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95" VIEWASTEXT>
   <PARAM NAME="AudioStream" VALUE="-1">
            .
            .
   <PARAM NAME="WindowlessVideo" VALUE="0">
</OBJECT>
```

12.1.3.2. Accessing the ActiveX.

Using its methods

```
<input type="button" value="Start" id="butStart" onclick="objWMPlay.play()">
<input type="button" value="Stop" id="butStop" onclick="objWMPlay.stop()">
```

Responding to events

```
<SCRIPT FOR="objWMPlay" EVENT="EndOfStream(lResult)" LANGUAGE="Jscript">
      alert("End of clip");
<SCRIPT>
```

12.1.3.3. To ask the user to install the ActiveX object in case s/he doesn't have already installed it, use the *CodeBase* attribute giving as a parameter the location and version of the ActiveX to install.

```
<OBJECT ID="objWMPlayer" classid="CLSID:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
CODEBASE="http://activex.microsoft.com/activex/controls/mplayer/en/
 nsmp2inf.cab#Version=6,4,5,715" VIEWASTEXT>
   <PARAM NAME="AutoStart" VALUE="True">
   <PARAM NAME="FileName" VALUE="test.avi">
</OBJECT>
```

12.1.3.4. Handling ActiveX Objects within Netscape Navigator

Netscape does not support ActiveX object and ignores the *Object* tag. Use the *Embed* element within the *Object* element to use an ActiveX as a plug-in. This code supports both browsers:

```
<OBJECT ID="objWMPlayer"
        classid="CLSID:22D6F312-B0F6-11D0-94AB-0080C74C7E95"
CODEBASE="http://activex.microsoft.com/activex/controls/mplayer/en/nsmp2inf
.cab#Version=6,4,5,715" type="application/x-oleobject" VIEWASTEXT>
   <PARAM NAME="AutoStart" VALUE="False">
   <PARAM NAME="FileName" VALUE="test.wav">
   <EMBED type="application/x-mplayer2"
         pluginspage="http://www.microsoft.com/Windows/MediaPlayer/"
         SRC="test.wav"
         name="objWMPlayer"
         autostart="0"
         showcontrols="1"
   </EMBED>
</OBJECT>
```

Also refresh the plug-in to no make the user to refresh the browser.

```
<script language="JavaScript">
   // If the browser is Netscape Navigator.
   if ( navigator.appName == "Netscape" )
   {
      // Refresh the plugins that were just installed.
      navigator.plugins.refresh();
    }
</script>
```

Because code to access the player is not compatible, it is better to disable the HTML controls set for Windows media player.

12.2. Displaying video

12.2.1. Using the *dynsrc* attribute of the *img* Element

This attribute is also supported by the Image and ImageButton server and HTML controls. It accepts only AVI files and cannot be controlled from client-side. It is not supported by Netscape.

```
<img dynsrc="clock.avi">
```

```
<asp:ImageButton id="ImageButton1" dynsrc="clock.avi" runat="server">
</asp:ImageButton>
```

12.2.2. Embedding Videos

Use the *embed* HTML element, which also is supported by Netscape. The user can use the media controls installed in his/her computer and can play multiple video formats.

```
<embed src="clock.avi" id="aviTest" hidden="false" height="200" width="200"
autostart="false" type="video/avi" loop="true"></embed>
```

12.2.3. Videos as ActiveX Objects

Use the same techniques than from Sound playing.

12.3. Animating Web pages.

12.3.1. Using the *Marque* HTML tag.

It is only supported by Explorer.

```
<marquee bgcolor="red" scrollamount="10" behavior="alternate" direction="right" loop="true">
   <h1 style="COLOR:Yellow">Notice Me</h1>
</marquee>
```

You can nest *Marque* elements.

```
<marquee scrollamount="10" behavior="alternate" direction="up" loop="true">
   <marquee scrollamount="5" behavior="alternate" direction="right"
   loop="true">
      <img src="circles.gif">
   </marquee>
</marquee>
```

12.3.2. Animating simple graphs.

12.3.2.1. Using animated GIF files.

12.3.2.2. Using plug ins such as Macromedia ShockWave or Windows Media Player

12.3.2.3. Using client-side scripts

12.3.2.4. Synchronize multimedia events in Internet Explorer using HTML+TIME, which is the Microsoft implementation of the SMIL standard.

12.3.2.5. HTML+TIME allows you to:

- Create different types of timelines on a Web page
- Specify the order, timing, and duration of events within a timeline
- Control the attributes, color, and position of elements in response to those timed events

13. Formatting Web Application Output

13.1. Style sheets store formatting information in a single location. There are 3 levels that affect formatting:

Levels of Style		
Level	Defined in	Applies to
Global	The style sheet file	All pages referencing the style sheet
Page	The page' s head element	All elements on the current page
Inline	The HTML element itself	Only the current element

Inline formatting take precedence over Page formatting, and Page formatting over Global formatting.

```
<HTML>
   <HEAD>
         <title>WebForm1</title>
<!-- (1) Style sheet reference. -->
         <LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css">
<!-- (2) Page-level style definition. -->
         <style>
         p {
         font-family: 'Comic Sans MS', Lucida Sans, sans-serif;
         font-size: medium;
         }
         </style>
   </HEAD>
   <body>
         <p>The alignment is from the style sheet.</p>
         <p>The font is from the style in the page's head element.</p>
<!-- (3) Inline style definition -->
         <p style="FONT-SIZE: large; FONT-STYLE: italic">The italic is from
         the inline style.</p>
   </body>
</HTML>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
<!-- (1) From Styles.css style sheet referenced in HEAD element. -->
p
```

Line to use a css file

CSS file utilized by Vstudio

```
{
   font-size: small;
   text-align: center;
}
```

In this example:

- The page level *font-size* never has a chance of being utilized.
- The css-file level *text-align*, the Page level *font-family* and the Inline level *font-style*, and *font-size* are additive and are utilized.

13.2. The advantages of using css-level style formatting are that formatting is maintained in just one location, and you can use several css files for different situations. For example one css file to display the page and another to print it.

13.3. Using the Style Builder to change the appearance.

13.4. Style classes allow you to apply the same formatting to different HTML elementson a Web form.

```
(1) Define the class
.emphasis
{
    font-style: italic;
}


(2) use the class
<p>This is a paragraph containing <span class="emphasis">emphasis</span>.</p>
<asp:TextBox ID="Text1" Runat="server" CssClass="emphasis">Some text
</asp:TextBox>
```

13.5. To apply formatting to a specific element in a form use its element ID. The following example applies formatting only to the elements IDs “Inserted” and “deleted”.

```
#inserted
{
      text-decoration: underline;
}

#deleted
{
      text-decoration: line-through;
}
```

13.6. Creating nested styles

The following styles specify different bullet types for nested, un-numerated lists:

```
UL LI    {
   list-style-type:  square ;
   }

UL LI LI {
   list-style-type:  disc;
   }

UL LI LI LI    {
   list-style-type:  circle;
   }
```

13.7. Changing Style Sheets at Run time.

13.7.1. To change automatically Style sheet for printing and another for displaying use the *media* attribute:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="Styles.css" media="screen">
<LINK REL="stylesheet" TYPE="text/css" HREF="Print.css" media="print">
```

13.7.2. Change the *href* attribute using a client-side attribute to change the style sheet at run time:

```
<HTML>
   <HEAD>
         <title>ChangeSheets</title>
     <LINK ID="ScreenStyle" REL="stylesheet" TYPE="text/css" HREF="Styles.css" media="screen">
     <LINK ID="PrintStyle" REL="stylesheet" TYPE="text/css" HREF="Print.css" media="print">

   <script id=clientEventHandlersJS language=jscript>
       function SwitchSheets()
       {
         // Switch between default and large-type style sheets.
         if (document.all["ScreenStyle"].getAttribute("HREF") == "Styles.css")
           document.all["ScreenStyle"].setAttribute ("HREF",  "BigType.css", 0);
         else
            document.all["ScreenStyle"].setAttribute ("HREF", "Styles.css", 0);
       }
   </script>
         </HEAD>
   <body MS_POSITIONING="FlowLayout">
      <FORM id="Form1" method="post" runat="server">
```

```
          <H2>Changing Style Sheets at Run-Time</H2>
          <P>This page first appears using the default style sheet.</P>
          <P><A onclick="SwitchSheets();" href="#">Click here</A> to switch between types </P>
        </FORM>
    </body>
</HTML>
```

13.7.3. Using Behaviors

Check the book.

13.8. Using XSL transformations

13.8.1. Summary of commands:

Basic XSL elements

Element	Attributes	Use to
xsl:stylesheet	*Version* *xmlns:xsl*	Identify the version of XSL being used. Version 1.0 is the current version at the time of this writing. Specify the prefixes for elements in the XSL file.
xsl:template	*match*	Define a template for an XML node.
xsl:apply-templates	*select*	Apply a template to the selected node.
xsl:value-of	*select*	Retrieve the value of an XML node or evaluate an XPath expression.
xsl:text		Include literal text or white space characters in the output.

Conditional XSL Elements

Element	Attributes	Use to
xsl:if	*test*	Test a Boolean expression and process the contained elements if the result is true.
xsl:choose		Test multiple conditions using contained xsl:when and xsl:otherwise elements. This is the equivalent of a Select Case (Visual Basic .NET) or switch statement (Visual C#).
xsl:when	*test*	Test one Boolean expression within a xsl:choose element, and then process the contained item and exit the xsl:choose element if the result is true. This is the equivalent of the Case (Visual Basic .NET) or case (Visual C#) statements.
xsl:otherwise		Process the contained item if none of the xsl:when elements within an xsl:choose element are true. This is the equivalent of the Case Else (Visual Basic .NET) or default statement (Visual C#).

13.8.2. Sample XML file to use

```
<cookbook>
  <title></title>
  <subtitle></subtitle>
  <introduction></introduction>
  <recipe>
      <title></title>
      <subtitle></subtitle>
      <description></description>
      <servings></servings>
      <ingredients>
          <ingredient>
              <name></name>
              <quantity></quantity>
          </ingredient>
          <ingredient>
              <name></name>
              <quantity></quantity>
          </ingredient>
      </ingredients>
      <instructions>
          <introduction></introduction>
          <step></step>
          <step></step>
          <step></step>
          <summary></summary>
      </instructions>
  </recipe>
</cookbook>
```

13.8.3. Simple formatting

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/cookbook">
    <xsl:apply-templates select="cookbook" />
```

```
    <h1><xsl:value-of select="title" /></h1>
    <h3><xsl:text>...</xsl:text><xsl:value-of select="subtitle" /></h3>
    <hr />
    <p><xsl:value-of select="introduction" /></p>
    <hr />
    <xsl:for-each select="recipe">
         <xsl:apply-templates select="." />
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="recipe">
    <h3><xsl:value-of select="title" /></h3>
    <p><xsl:value-of select="description" /></p>
    <p><b><xsl:text>Serves: </xsl:text></b>
    <i><xsl:value-of select="servings" /></i></p>
    <xsl:apply-templates select="ingredients" />
    <xsl:apply-templates select="instructions" />
  </xsl:template>

  <xsl:template match="ingredients">
    <ul>
      <xsl:for-each select="ingredient">
         <li><xsl:value-of select="name" />
         <xsl:text>  </xsl:text>
         <b><xsl:value-of select="quantity" /></b></li>
      </xsl:for-each>
    </ul>
  </xsl:template>

  <xsl:template match="instructions">
    <p><xsl:value-of select="introduction" /></p>
    <ol>
      <xsl:for-each select="step">
         <li><xsl:value-of select="." /></li>
      </xsl:for-each>
    </ol>
    <p><xsl:value-of select="summary" /></p>
  </xsl:template>

</xsl:stylesheet>
```

13.8.4. Formatting with table of contents and hyperlinks

```
<xsl:template match="/cookbook">
   <h1><xsl:value-of select="title" /></h1>
   <h3><xsl:text>...</xsl:text><xsl:value-of select="subtitle" /></h3>
   <hr />
   <p><xsl:value-of select="introduction" /></p>
   <hr />
   <!-- Display table of contents -->
<h4>Contents</h4>
<xsl:apply-templates mode="contents" select="recipe" />
   <xsl:for-each select="recipe">
      <xsl:apply-templates select="." />
   </xsl:for-each>
</xsl:template>

<xsl:template match="recipe">
      <!-- Create location links -->
      <xsl:element name="a">
            <xsl:attribute name="name">
                  <xsl:value-of select="generate-id(title)" />
            </xsl:attribute>
            <h3><xsl:value-of select="title" /></h3>
      </xsl:element>
      <p><xsl:value-of select="description" /></p>
      <p><b><xsl:text>Serves: </xsl:text></b>
      <i><xsl:value-of select="servings" /></i></p>
      <xsl:apply-templates select="ingredients" />
      <xsl:apply-templates select="instructions" />
</xsl:template>

SEE INGREDIENTS AND INSTRUCTIONS MATCHES FROM THE FORMER XSL

<xsl:template mode="contents" match="recipe">
      <!-- Create table of contents links -->
      <xsl:element name="a">
            <xsl:attribute name="href">
                  <xsl:text>#</xsl:text>
                  <xsl:value-of select="generate-id(title)" />
            </xsl:attribute>
```

Generates a unique Hash ID based in the same string.

```
            <xsl:value-of select="title" />
      </xsl:element>
      <br />
</xsl:template>
```

Resulting HTML:

```
HEADER WITH TABLE OF CONTENTS

<h1>Stewing in Florida </h1>
<h3>...It's Hot and I'm Hungry</h3>
<hr />
<p>
   I learned about cooking from my father camping just off the Tamiami trail on our
   monthly jaunts to the Everglades. I don't know who ate better, us or the mosquitoes,
   but I do remember enjoying the time, cooking what we had, and trying to keep the
   bugs out of the pot.
</p>
<hr />
<h4>Contents</h4>
<a href="#XSLTtitle120123120120">Cooter Stew</a><br />
<a href="#XSLTtitle120124120120">Shmondue</a><br />
<a href="#XSLTtitle120125120120">Test 3</a><br />
<a href="#XSLTtitle120126120120">Test 2</a><br />
<a href="#XSLTtitle120127120120">Test 1</a><br />

RECIPE

<a name="XSLTtitle120123120120"> <h3>Cooter Stew</h3> </a>
<p>
  You can't get cooter any more, and you shouldn't try -- they're protected and
  they don't really taste very good. However, you can substitute just about any
  white meat for the cooter in this recipe. Chicken or pork will do.
</p>
<p><b>Serves: </b><i>4 to 6</i></p>

INGREDIENTS

<ul>
  <li>Cooter   <b>2-3 pounds</b></li>
  <li>Potatoes <b>3 pounds</b></li>
  <li>Tomatoes <b>2 pounds</b></li>
  <li>Onions   <b>3 cups</b></li>
  <li>Okra     <b>2 cups, more if you like okra</b></li>
  <li>Salt     <b>1 TBSP</b></li>
  <li>Pepper   <b>2 tsp</b></li>
  <li>Oil      <b>For frying</b></li>
</ul>

INSTRUCTIONS

<p>
  Cooter is what we call gopher turtles -- which aren't gophers and aren't
  really turtles, either. They are a kind of tortose that lives in the sandy
  pine woods and if you see one you shouldn't bother it. Instead, use boned
  chicken or pork "country style" ribs for this recipe.
</p>
<ol>
  <li>Chop the meat and vegetables into 1-inch chunks.</li>
  <li>Saute onions in oil till translucent.</li>
  <li>Season meat and add to pan with onion. Saute 10 minutes.</li>
  <li>Add meat, vegetables, and salt to large pot and add water to cover.</li>
  <li>Heat till boiling, then reduce heat and simmer for 1 hour.</li>
  <li>If you were using real cooter, you'd have to remove skim the scum from pot
      every 10 minutes or so and add 1 hour to cook time.</li>
</ol>
<p>Serve with Saltine crackers and Red Devil hot sauce.</p>

ANOTHER RECIPE...

<a name="XSLTtitle120124120120"><h3>Shmondue</h3></a>
<p>
  We came up with this when we were really bored. First we decided to see
  what would happen if you put a marshmellow in the microwave on high. Then we
  decided to see if we could make microwave s'mores. Finally, we settled on something
  we could actually eat.
</p>
     .
     .
     .
```

13.8.5. Sorting items

```
<h4>Contents</h4>
<xsl:apply-templates mode="contents" select="recipe">
   <xsl:sort select="title" order="ascending" />
</xsl:apply-templates>
```

13.8.6. Performing conditional tasks. Uses the Xpath function "count"

```
<xsl:if test="count(recipe) > 4">
   <h4>Contents</h4>
   <xsl:apply-templates mode="contents" select="recipe" />
</xsl:if>
```

14. Providing Help

14.1. Displaying ToolTips

14.1.1. Explorer displays the HTML controls's title as a ToolTip. ASP.NET controls include a *TootTip* property that is converted to HTML control's *title* (which Explorer displays as a ToolTip). DropDownList and ListBox don't include *ToolTip* property.

```
<asp:Button id="butOK" runat="server" Text="OK" ToolTip="Submits purchase."></asp:Button>
<INPUT type="reset" value="Cancel" title="Cancels purchase.">
```

14.1.2. Displaying Help as Web Forms or HTML documents.
To display Help in a separate window, use the client-side window object' s *open* method, as shown by the following hyperlink:

```
<a href="#" onclick="window.open('topic1.aspx', 'helpwin').focus()">
Click here for help.</a>
```

- `href="#"` links to the current location in the Web application
- `'topic1.aspx'` is the help topic to display
- `'helpwin'` names the new window so that subsequent Help files are directed to the same window.

To close the Help window, you can place a script on it to close it:

```
<a href="#" onclick="window.close()">Click here to close help.</a>
```

14.1.3. Displaying HTML Help
The *window* object's *showHelp* method uses the HTML Help Viewer (hh.exe) to display either a Web form or an HTML file. This example displays the topic 1 of the web form.

```
<a href="#" onclick="window.showHelp('Topic1.aspx')">Show Help</a>
```

The same method can be used to display compiled HTML files (.chm) created with the HTML Help WorkShop utility. The file has to reside in the client's computer, and you have to pass as a parameter the file name and topic to display

```
<a href="#" onclick="window.showHelp('c:\\Help\\HelpSample.chm')">
Show Help </a><br>
```

14.1.4. To invite the user to download the help file use the following link

```
<a href="HelpSample.chm">Click here</a> to download the sample Help file and
then save the file to the C:\Help folder on your computer.
```

- After clicking, the browser asks the user the place to download the file.

14.1.5. To displays a Help topic in the browser from the Internet, use the following link in the Help file.

```
<a href="ms-its:http://www.mycompany.com/Help/HelpSample.chm::/topic1.htm" target="HelpWin">
Show Help</a>
```

- The link uses the ms-its protocol available since Explorer 4.0

14.1.6. To display context-sensitive help, make available an event in the HTML page that calls a topic using the *window.showHelp* function:

- An HTML control would provide the topic to display

```
<INPUT onfocus="ShowContextHelp('Topic2.htm')" type="text">
```

- The ShowContext Jscript function would display the help topic

```
window.showHelp "c:\help\HelpSample.chm::/" & Topic
```

14.1.7. Using the HTML Help workshop

15. Globalizing Web Applications

15.1. There are 3 ways to structure an application for Globalization, from the simplest to the most complex they are:

Globalization Approaches		
Approach	Description	Best for
Detect and redirect	Create a separate Web application for each supported culture, and then detect the user' s culture and redirect the request to the appropriate application.	Applications with lots of text content that requires translation and few executable components.

Globalization Approaches		
Run-time adjustment	Create a single Web application that detects the user' s culture and adjusts output at run time using format specifiers and other tools.	Simple applications that present limited amounts of content.
Satellite assemblies	Create a single Web application that stores culture-dependent strings in resource files that are compiled into satellite assemblies. At run time, detect the user' s culture and load strings from the appropriate assembly.	Applications that generate content at run time or that have large executable components.

15.2. To use any of the approaches you need to identify the user's culture. Use the *Request*'s *UserLanguages* property. The first element of the array is the language.

```
Imports System.Globalization
    .
private void Page_Load(object sender, System.EventArgs e)
{
   // Get the user's preferred language.
   string sLang = Request.UserLanguages[0];
   // Get the first two characters of language.
   sLang = sLang.Substring(0, 2);
   // Redirect user based on his/her culture.
   switch (sLang)
   {
         case "en":
               // Use US site.
               Response.Redirect("http://www.contoso.com/usa");
               break;
         case "es":
               // Use Spanish site.
               Response.Redirect("http://www.contoso.com/es");
               break;
         case "de":
               // Use German site.
               Response.Redirect("http://www.contoso.com/de");
               break;
         case "zh":
               // Use Chinese site.
               Response.Redirect("http://www.contoso.com/zh");
               break;
         default:
               // Use US site.
               Response.Redirect("http://www.contoso.com/usa");
               break;
         }
   }
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Also you can set the *globalization* element of the Web.config file, this will set formatting of dates, currency, and numbers.
Using Web.config to set the culture creates a static association between the application and a specific culture.

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8" culture="ar-SA" />
```

15.3. Adjusting to Current Culture at Run Time

You can set the culture dynamically at run time using the *Thread* class' s *CurrentCulture* property, as shown here:

```
using System.Globalization;
using System.Threading;

private void Page_Load(object sender, System.EventArgs e)
{
      // Get the user's preferred language.
   sLang = Request.UserLanguages[0];
     // Set the thread's culture to match the user culture.
   Thread.CurrentThread.CurrentCulture = new CultureInfo(sLang);

   // **** Using the culture information ****
     // Display the culture's native name in the heading.
   head1.InnerHtml = Thread.CurrentThread.CurrentCulture.NativeName;
     // Display Date, Currency, and Numeric strings formatted for the culture.
   Label1.Text = DateTime.Now.ToString("F");
   Label2.Text = 1234567890.ToString("C");
   Label3.Text = 1234567890.ToString("N");
     // If culture is Arabic or Hebrew, use right-to-left layout.
   if ((sLang == "ar") ||(sLang == "he"))
    {
      Panel1.Attributes.Add("dir", "rtl");
    }
}
```

15.4. Creating and using Satellite Assemblies.

Steps:

15.4.1. Enable HTML elements for Resources.

Set the *id* and *runat* attributes of all the user-interface elements that require translations. ASP.NET controls include them by default. The following file shows the new attributes in bold type and you will use their *InnerHTML* property to change their text.

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbSatelliteSnippet.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
   <HEAD>
         <title>WebForm1</title>
   </HEAD>
   <body>
         <form id="Form1" method="post" runat="server">
               <h1 id="head1" runat="server">Sample heading</h1>
               <p id="p1" runat="server">Some introductory text.</p>
               <span id="sp1" runat="server">Enter a number: </span>
               <asp:TextBox ID="txtCurrency" Runat="server" />
               <br>
               <span id="sp2" runat="server">Amount in local currency: </span>
               <asp:Label ID="lblCurrency" Runat="server" />
               <br>
               <br>
               <asp:Button ID="butOK" Text="OK" Runat="server" />
               <input id="butCancel" runat="server" type="button"
                 onclick="txtCurrencty.value=''" value="Cancel">
         </form>
   </body>
</HTML>
```

15.4.2. Create resource files

15.4.2.1. Use Vstudio, from the Project menu choose Add New Item and the select Assembly Resource File.

15.4.2.2. Create a translated field for each element to be translated. Assign them unique IDs for the whole application, a good alternative is to use the naming format *formname.elementname*, example: webform1.butok

15.4.2.3. You can create three types of resource files

Resource File Types for Satellite Assemblies			
Type	Named	Compiled into	Provides
Fallback	file.resx. For example: **strings.resx**	The executable assembly.	User-interface strings when culture is not specified or not recognized by the Web application.
Language-specific	file.langcode.resx. For example: **strings.es.resx**	Resource-only assembly stored in a subfolder of the bin folder identified by language code. For example: **bin\es**	Translated strings for cultures using a particular language.
Culture-specific	file.langcode-regioncode.resx. For example: **strings.es-MX.resx**	Resource-only assembly stored in a subfolder of the bin folder identified by the language and region code. For example: **bin\es-MX**	Translated strings for cultures using a specific dialect of a language.

15.4.3. Implement code similar to this example to display the resource strings

```
using System.Globalization;
using System.Threading;
using System.Resources;   //To use the Resource Manger class

// Load resources stored in the Strings assembly file.
protected ResourceManager gStrings = new
  ResourceManager("csSatelliteSnippet.strings", typeof(WebForm1).Assembly);

private void Page_Load(object sender, System.EventArgs e)
{
   if (!IsPostBack)
   {
         // Get the user's perferred language.
       string sLang = Request.UserLanguages[0];
         // Set the thread's culture for formatting, comparisons, etc.
       Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(sLang);
         // Set the thread's UICulture to load resources from satellite assembly.
       Thread.CurrentThread.CurrentUICulture = new CultureInfo(sLang);

         // Get strings from resource file.
       head1.InnerHtml = gStrings.GetString("webform1.head1");
       p1.InnerHtml = gStrings.GetString("webform1.p1");
       sp1.InnerHtml = gStrings.GetString("webform1.sp1");
       sp2.InnerHtml = gStrings.GetString("webform1.sp2");
       butOK.Text = gStrings.GetString("webform1.butOK");
       butCancel.Value = gStrings.GetString("webform1.butCancel");
   }
```

The first parameter is the namespace plus the resource base name without culture code. The second parameter is an object representing the current assembly

Tries to get an specific cuture (es-MX). If not, then tries to get a neutral culture (es), finally brings the fallback resources

Assign the strings retrieved to *Text*, *Value*, or *inerHTML* properties of the Web form's elements.

```
}

//To make desicions based in the culture
private void butOK_Click(object sender, System.EventArgs e)
{
      // (1) Create culture variables for formatting.
      CultureInfo Europe = new CultureInfo("fr-FR");
      CultureInfo USA = new CultureInfo("en-US");

      // (2) Perform conversions based on current user-interface culture.
      switch (Thread.CurrentThread.CurrentCulture.Name)
      {
            case "en-US":
                  lblCurrency1.Text = DestAmount.ToString("C", USA);
                  break;
            case "fr-FR":
                  lblCurrency1.Text = DestAmount.ToString("C", Europe);
                  break;
            default:
                  lblCurrency1.Text = DestAmount.ToString("C", USA);
                  break;
            }
   }
```

HTML control

15.5. Globalization Issues

15.5.1. General Programming issues

15.5.1.1. When implementing custom techniques for sorting, use the *System.Globalization* namespace's *CompareInfo* class to perform culturally aware strings comparisons.

15.5.1.2. Use the *Convert* class within an exception-handling structure to test if a value is numeric.

15.5.1.3. Store strings in a resource file to retrieve as necessary instead of trying to use concatenating functions.

15.5.1.4. Getting sub-strings can fail in languages that use symbols.

15.5.2. When creating Web forms that use non-ASCII characters, save the file using the UTF-8 encoding with a signature. Including the signature allows ASP.NET to automatically detect the file' s encoding. If you don't't include the signature, specify the *fileEncoding* attribute in the Web.config file

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8" fileEncoding="utf-8" />
```

15.5.3. Using other encodings.

15.5.3.1. To detect when a request has a specific encoding, use the *ContentEncoding* property of the *Request* object to detect a request with a specific encoding. Here the code displays the request's encoding name.

```
Response.Write(Request.ContentEncoding.WebName);
```

15.5.3.2. To set the *ContentEncoding* property of the Response object to set a specific encoding. You can do it in two ways:

- Using the *Encoding* class of the *System.Text* name space
- Using the value returned from the Request object's *ContentEncoding* property.

```
//Create a response using the shift_JIS character encoding
using System.Text;
private void Page_Load(object sender, System.EventArgs e)
{
   Response.ContentEncoding = Encoding.GetEncoding("shift_JIS");
   Response.Write("This response uses the character encoding: ");
   Response.Write(Response.ContentEncoding.WebName);
}
```