FREE eBook

LEARNING ASP.NET

Free unaffiliated eBook created from
Stack Overflow contributors.

#asp.net

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: asp-net

It is an unofficial and free ASP.NET ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ASP.NET.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with ASP.NET

Remarks

ASP.NET is a collection of technologies within the .NET Framework that are targeted towards web application development. These technologies consist of:

- WebForms: A RAD style development platform using web controls.
- MVC: A Model View Controller development platform.
- SignalR: A real-time messaging platform for client/server messaging.
- Razor: A front-end markup language you can embed server-side commands with.
- WebAPI: A platform for building REST API style applications.

Examples

Installation or Setup

By default, all the required libraries for build ASP.NET applications are included during the installation of Visual Studio. If a newer version of ASP.NET is released that was not included with Visual Studio, you can download the appropriate SDK library from Microsoft, which will include all the necessary libraries for that version.

Similarly, the Windows operating system comes pre-installed with a more recent version of ASP.NET and is automatically registered with IIS for configuration and execution. Similarly, if a newer version of ASP.NET becomes available, you can install the SDK for the version you need and then use the `aspnet_regiis` tool to register the framework with IIS for use.

It should be also noted that for server deployments, there also exists a ASP.NET SDK Redistributable package. This version is a streamlined version of the SDK, with just the essential libraries and does not have the tools and integrations with Visual Studio in it.

ASP.NET Overview

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework.

You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic, C#, JScript .NET, and J#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on.

ASP.NET includes:

- A page and controls framework

- The ASP.NET compiler
- Security infrastructure
- State-management facilities
- Application configuration
- Health monitoring and performance features
- Debugging support
- An XML Web services framework
- Extensible hosting environment and application life cycle management
- An extensible designer environment

Hello World with OWIN

Use the packet manager to install Microsoft.Owin.SelfHost

```
install-packet Microsoft.Owin.SelfHost
```

Code for a bare minimum HelloWorld web application running from a console window:

```
namespace HelloOwin
{
    using System;
    using Owin;

    class Program
    {
        static readonly string baseUrl = "http://localhost:8080";

        static void Main(string[] args)
        {
            using (Microsoft.Owin.Hosting.WebApp.Start<Startup>(baseUrl))
            {
                Console.WriteLine("Prease any key to quit.");
                Console.ReadKey();
            }
        }


    }

    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.Run(ctx =>
            {
                return ctx.Response.WriteAsync("Hello World");
            });
        }
    }
}
```

Simple Intro of ASP.NET

Asp.net is web application framework developed by Microsoft to build dynamic data-driven Web

Application and WebServices.

Asp.net is basically a subset of wider .NET framework. A framework is nothing but a collection of classes.

In .NET Framework you can build Console application. Web Application, Window Application, Mobile Application. So for web application ASP.net is being used.

ASP.NET is the successor to classic ASP (Active Server Page.)

What is Web Application?

A web application is an application that is accessed by users using a web browser such as:

- Microsoft Internet Explorer.
- Google Chrome
- Mozilla FireFox
- Apple safari

Read Getting started with ASP.NET online: https://riptutorial.com/asp-net/topic/836/getting-started-with-asp-net

Chapter 2: Asp Web Forms Identity

Examples

Getting Started

Getting Started

Install NuGet packages:

1. **Microsoft.AspNet.Identity.EntityFramework**
2. **Microsoft.AspNet.Identity.Core**
3. **Microsoft.AspNet.Identity.OWIN**

Register action - Account controller

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser() { UserName = model.UserName };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInAsync(user, isPersistent: false);
            return RedirectToAction("Index", "Home");
        }
        else
        {
            AddErrors(result);
        }
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Log-in action - SignInAsync method

```
private async Task SignInAsync(ApplicationUser user, bool isPersistent)
{
    AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie);

    var identity = await UserManager.CreateIdentityAsync(
       user, DefaultAuthenticationTypes.ApplicationCookie);

    AuthenticationManager.SignIn(
       new AuthenticationProperties() {
```

```
            IsPersistent = isPersistent
        }, identity);
}
```

Log off

```
// POST: /Account/LogOff
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
    AuthenticationManager.SignOut();
    return RedirectToAction("Index", "Home");
}
```

Read Asp Web Forms Identity online: https://riptutorial.com/asp-net/topic/9146/asp-web-forms-identity

Chapter 3: ASP.NET - Basic Controls

Syntax

- <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" / > <asp:TextBox ID="txtstate" runat="server">
- </asp:TextBox> <asp:CheckBox ID= "chkoption" runat= "Server"> </asp:CheckBox> <asp:RadioButton ID= "rdboption" runat= "Server"> </asp: RadioButton>
- <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="ListBox1_SelectedIndexChanged"> </asp:ListBox>
- <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged"> </asp:DropDownList>
- <asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged"> </asp:RadioButtonList>
- <asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged"> </asp:CheckBoxList>
- <asp:BulletedList ID="BulletedList1" runat="server"> </asp:BulletedList>
- <asp:HyperLink ID="HyperLink1" runat="server"> HyperLink </asp:HyperLink> <asp:Image ID="Image1" runat="server">

Examples

Text Boxes and Labels

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

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

Common Properties of the Text Box and Labels:

Properties	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLIne creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box.

Properties	Description
MaxLength	The maximum number of characters that can be entered into the text box.
Wrap	It determines whether or not text wraps automatically for multi-line text box; default is true.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user can change the text.
Columns	The width of the text box in characters. The actual width is determined based on the font that is used for the text entry.
Rows	The height of a multi-line text box in lines. The default value is 0, means a single line text box.

The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

```
<asp:CheckBox ID= "chkoption" runat= "Server"> </asp:CheckBox>
```

Basic syntax of radio button:

```
<asp:RadioButton ID= "rdboption" runat= "Server"> </asp: RadioButton>
```

Common properties of check boxes and radio buttons:

Properties	Description
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

List Controls

ASP.NET provides the following controls

- Drop-down list
- List box
- Radio button list
- Check box list
- Bulleted list

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

```
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Basic syntax of drop-down list control:

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Common properties of list box and drop-down Lists:

Properties	Description
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.
Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.
SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string (""").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

Common properties of each list item objects:

Properties	Description
Text	The text displayed for the item.
Selected	A string value associated with the item.
Value	Indicates whether the item is selected.

It is important to notes that:

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of radio button list:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"
   OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

Basic syntax of check box list:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
   OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Common properties of check box and radio button lists:

Properties	Description
RepeatLayout	This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

Bulleted lists and Numbered lists

The bulleted list control creates bulleted lists or numbered lists. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of a bulleted list:

```
<asp:BulletedList ID="BulletedList1" runat="server">
</asp:BulletedList>
```

Common properties of the bulleted list:

Properties	Description
BulletStyle	This property specifies the style and looks of the bullets, or numbers.
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical.
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

HyperLink Control

The HyperLink control is like the HTML element.

Basic syntax for a hyperlink control:

```
<asp:HyperLink ID="HyperLink1" runat="server">
   HyperLink
</asp:HyperLink>
```

It has the following important properties:

Properties	Description
ImageUrl	Path of the image to be displayed by the control.
NavigateUrl	Target link URL.
Text	The text to be displayed as the link.
Target	The window or frame which loads the linked page.

Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

```
<asp:Image ID="Image1" runat="server">
```

It has the following important properties:

Properties	Description
AlternateText	Alternate text to be displayed in absence of the image.
ImageAlign	Alignment options for the control.
ImageUrl	Path of the image to be displayed by the control.

Read ASP.NET - Basic Controls online: https://riptutorial.com/asp-net/topic/6444/asp-net---basic-controls

Chapter 4: ASP.NET - Managing State

Examples

View State

The following example demonstrates the concept of storing view state. Let us keep a counter, which is incremented each time the page is posted back by clicking a button on the page. A label control shows the value in the counter.

The markup file code is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="statedemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

   <head runat="server">
      <title>
         Untitled Page
      </title>
   </head>

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

         <div>
            <h3>View State demo</h3>

            Page Counter:

            <asp:Label ID="lblCounter" runat="server" />
            <asp:Button ID="btnIncrement" runat="server" Text="Add Count"
onclick="btnIncrement_Click" />
         </div>

      </form>
   </body>

</html>
```

The code behind file for the example is shown here:

```
public partial class _Default : System.Web.UI.Page
{
   public int counter
   {
      get
      {
         if (ViewState["pcounter"] != null)
         {
```

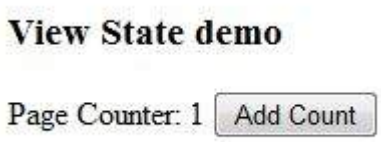
```
            return ((int)ViewState["pcounter"]);
         }
         else
         {
            return 0;
         }
      }

      set
      {
         ViewState["pcounter"] = value;
      }
   }

   protected void Page_Load(object sender, EventArgs e)
   {
      lblCounter.Text = counter.ToString();
      counter++;
   }
}
```

It would produce the following result:

View State Demo



Read ASP.NET - Managing State online: https://riptutorial.com/asp-net/topic/6296/asp-net---managing-state

Chapter 5: ASP.NET - User Controls

Introduction

User controls are containers which can be populated with HTML markup & server controls with code-behind in the same way as ASPX page. They're treated as reusable smaller units of a page, so they can't run as stand-alone pages and must not having **html**, **body** or **form** HTML elements in them.

Examples

Introduction of User Controls

User controls are made for reusability across ASP.NET pages, similar to master pages. Instead of sharing base page layout, user controls share group of HTML/ASP.NET built-in server controls or a specific form layout, e.g. comment submission or guest notes.

A user control can contain both HTML controls and ASP.NET server controls, including client-side scripts.

The user controls usually include `Control` directive on top of its definition:

```
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
```

Like ASPX page, user controls consists of markups which can be associated with a code behind file to perform certain events and tasks, therefore all HTML tags available on ASPX page can be used on user controls except `<html>`, `<body>` and `<form>` tags.

Here is an example for simple user control markup:

```
<%-- UserControl.ascx --%>
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
<div>
    <asp:Label ID="Label1" runat="server" />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Click Here" OnClick="Button1_Click" />
</div>
```

Code-behind example:

```
// UserControl.ascx.cs
public partial class UserControl : System.Web.UI.UserControl
{
    protected void Button1_Click(Object sender, EventArgs e)
    {
        Label1.Text = "Hello World!";
    }
}
```

Before a user control inserted in ASPX page, `Register` directive should declared on top of the page referencing the user control with its source URL, tag name & tag prefix.

```
<%@ Register Src="UserControl.ascx" TagName="UserControl" TagPrefix="uc" %>
```

Afterwards, you can place user control inside ASPX page like ASP.NET built-in server control:

```
<uc:UserControl ID="UserControl1" runat="server" />
```

Creating User Control Instance Programmatically

If you want to instantiate an instance of user control inside ASPX code behind page, you need to write user control declaration on `Page_Load` event as follows:

```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
        Control control1 = LoadControl("UserControl.ascx");
        Page.Controls.Add(control1);
    }
}
```

Note that the user control ASCX file should be already created when executing LoadControl method.

Another way known to declare user controls programatically is using `PlaceHolder`:

```
public partial class Default : System.Web.UI.Page
{
    public PlaceHolder Placeholder1;
    protected void Page_Load(Object sender, EventArgs e)
    {
        Control control1 = LoadControl("UserControl.ascx");
        Placeholder1.Controls.Add(control1);
    }
}
```

Depending on your need, `PlaceHolder` places user controls on a container storing all server controls dynamically added into the page, where `Page.Controls` directly inserts user control inside the page which more preferred for rendering HTML literal controls.

Adding Custom Properties for User Control

Like standard ASP.NET built-in server controls, user controls can have properties (attributes) on its definition tag. Suppose you want to add color effect on `UserControl.ascx` file like this:

```
<uc:UserControl ID="UserControl1" runat="server" Color="blue" />
```

At this point, custom attributes/properties for user controls can be set by declaring properties

inside user control's code behind:

```
private String _color;
public String Color
{
    get
    {
        return _color;
    }
    set
    {
        _color = value;
    }
}
```

Additionally, if you want to set default value on a user control property, assign the default value inside user control's constructor method.

```
public UserControl()
{
    _color = "red";
}
```

Then, user control markup should be modified to add color attribute as following example:

```
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
<div>
    <span style="color:<%= Color %>"><asp:Label ID="Label1" runat="server" /></span>
    <br />
    <asp:Button ID="Button1" runat="server" Text="Click Here" OnClick="Button1_Click" />
</div>
```

Read ASP.NET - User Controls online: https://riptutorial.com/asp-net/topic/6773/asp-net---user-controls

Chapter 6: ASP.NET - Validators

Syntax

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

- **RangeValidator Control:**

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

- **CompareValidator Control:** <asp:CompareValidator ID="CompareValidator1" runat="server" ErrorMessage="CompareValidator"> </asp:CompareValidator>

- **CustomValidator:**

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

  </asp:CustomValidator>

- **Validation Summary:** <asp:ValidationSummary ID="ValidationSummary1" runat="server" DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />

Examples

Validation controls

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

RequiredFieldValidator Control

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

The syntax of the control is as given:

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

</asp:RequiredFieldValidator>
```

RangeValidator Control

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

It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date,
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

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

</asp:RangeValidator>
```

CompareValidator Control

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

It has the following specific properties:

Properties	Description
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
ValueToCompare	It specifies the comparison operator, the available values are: Equal,

Properties	Description
	NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

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

</asp:CompareValidator>
```

RegularExpressionValidator

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

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

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

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

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

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

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

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

The syntax of the control is as given:

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

</asp:RegularExpressionValidator>
```

Validation Summary

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

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

ShowSummary : shows the error messages in specified format.

ShowMessageBox : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
```

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

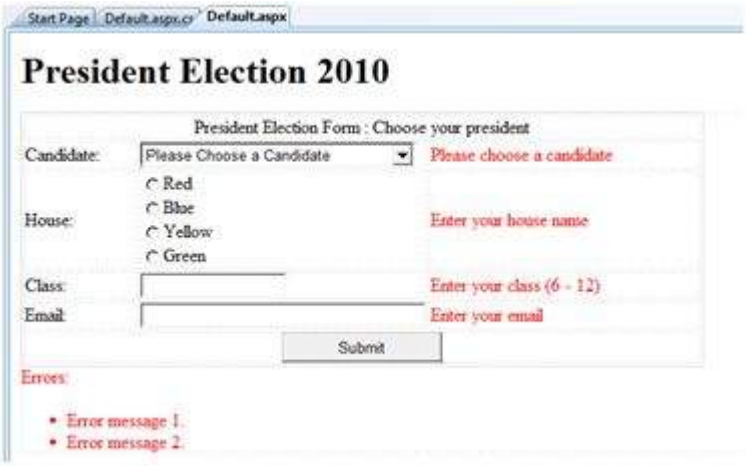
Validation Groups

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

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their ValidationGroup property.

Example The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:



The content file code is as given:

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

   <table style="width: 66%;">

      <tr>
         <td class="style1" colspan="3" align="center">
         <asp:Label ID="lblmsg"
            Text="President Election Form : Choose your president"
            runat="server" />
         </td>
      </tr>

      <tr>
         <td class="style3">
            Candidate:
         </td>

         <td class="style2">
            <asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
               <asp:ListItem>Please Choose a Candidate</asp:ListItem>
               <asp:ListItem>M H Kabir</asp:ListItem>
               <asp:ListItem>Steve Taylor</asp:ListItem>
```

```
            <asp:ListItem>John Abraham</asp:ListItem>
            <asp:ListItem>Venus Williams</asp:ListItem>
         </asp:DropDownList>
      </td>

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

   <tr>
      <td class="style3">
         House:
      </td>

      <td class="style2">
         <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
            <asp:ListItem>Red</asp:ListItem>
            <asp:ListItem>Blue</asp:ListItem>
            <asp:ListItem>Yellow</asp:ListItem>
            <asp:ListItem>Green</asp:ListItem>
         </asp:RadioButtonList>
      </td>

      <td>
         <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
            ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
         </asp:RequiredFieldValidator>
         <br />
      </td>
   </tr>

   <tr>
      <td class="style3">
         Class:
      </td>

      <td class="style2">
         <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
      </td>

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

   <tr>
      <td class="style3">
         Email:
      </td>

      <td class="style2">
         <asp:TextBox ID="txtemail" runat="server" style="width:250px">
```

```
            </asp:TextBox>
         </td>

         <td>
            <asp:RegularExpressionValidator ID="remail" runat="server"
               ControlToValidate="txtemail" ErrorMessage="Enter your email"
               ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
            </asp:RegularExpressionValidator>
         </td>
      </tr>

      <tr>
         <td class="style3" align="center" colspan="3">
            <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
               style="text-align: center" Text="Submit" style="width:140px" />
         </td>
      </tr>
   </table>
   <asp:ValidationSummary ID="ValidationSummary1" runat="server"
      DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
   if (Page.IsValid)
   {
      lblmsg.Text = "Thank You";
   }
   else
   {
      lblmsg.Text = "Fill up all the fields";
   }
}
```

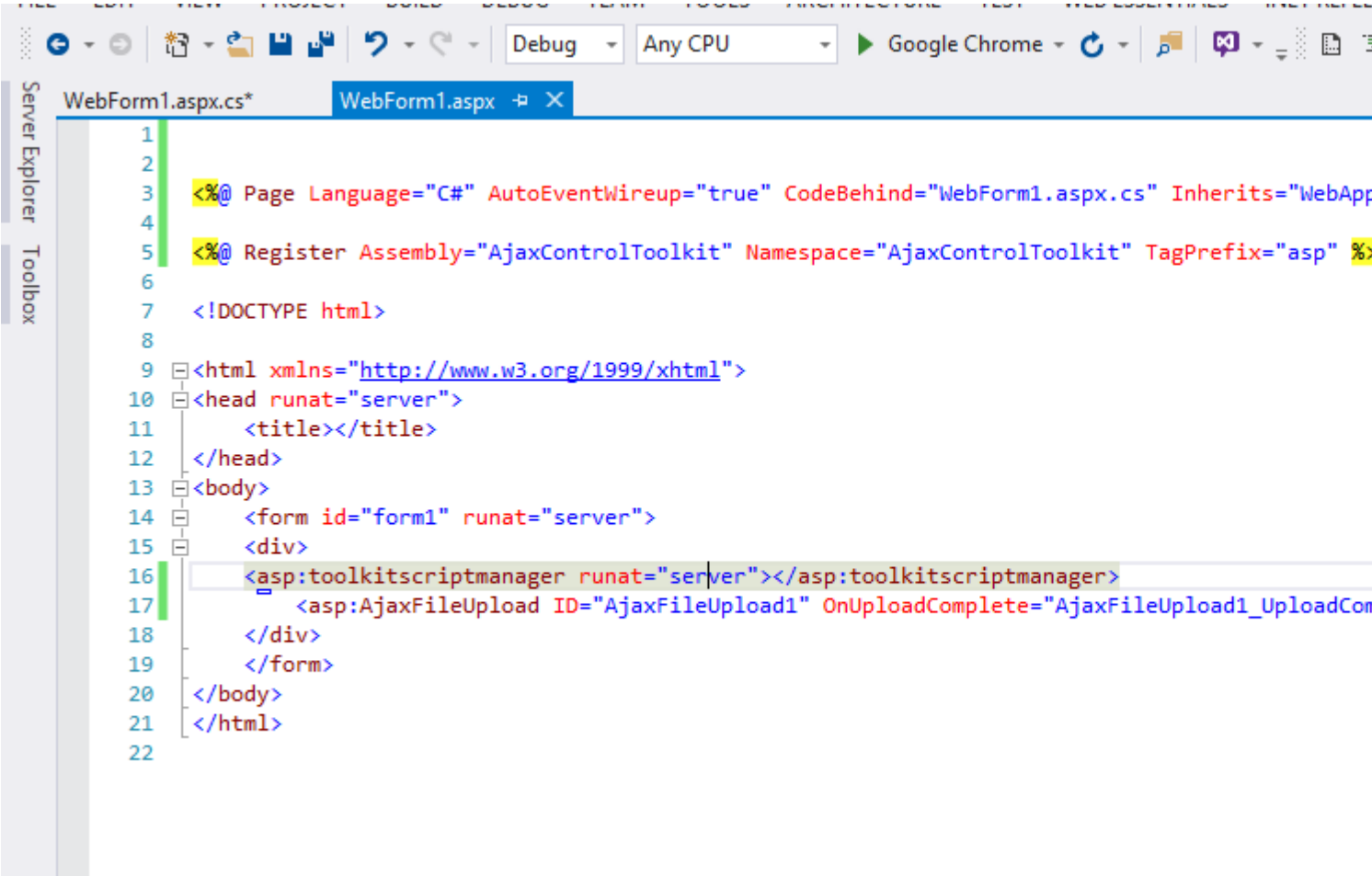
Read ASP.NET - Validators online: https://riptutorial.com/asp-net/topic/6180/asp-net---validators

Chapter 7: Asp.net Ajax Controls

Examples

FileUpload Ajax Toolkit Control

1. Add a reference of `AjaxToolkitControl.dll` into your project.
2. Then drag and drop `Toolkit Script Manager` and `AjaxFileUpload` Control from Visual Studio Toolbox window to your `.aspx` page like this :



3. use this code on your aspx.cs file

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

namespace WebApplication1
{
    1 reference
    public partial class WebForm1 : System.Web.UI.Page
    {
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        0 references
        protected void AjaxFileUpload1_UploadComplete(object sender, AjaxControlToolkit.AjaxFileUploadEve
        {
            string fileName = Path.GetFileName(e.FileName);
            AjaxFileUpload1.SaveAs(Server.MapPath("~/Uploads/" + fileName));
        }
    }
}
```

4. Make sure you have created folder named as **Uploads** in your project root directory.

Read Asp.net Ajax Controls online: https://riptutorial.com/asp-net/topic/7164/asp-net-ajax-controls

Chapter 8: ASP.NET Caching

Examples

Data Cache

ASP.Net exposes Cache API to store data in the cache for retrieval later.

Getting Started

Store string

```
   Cache["key"]="value";
```

Retrieve string

```
var value="";
if (Cache["key"] != null)
   value = Cache["key"].ToString();
```

You can also use the **Add** or the **Insert** methods.

```
protected void Page_Load( object sender, EventArgs e)
{
    if ( this.IsPostBack )
    {
        label1.Text + = "Page is posted back";
    }
    else
    {
        label1.Text + = "Page is created";
    }

    if ( Cache [ "item"] == null )
    {
        label1.Text + = "New item is created";
        DateTime item = DateTime.Now;
        label1.Text + = "Item is stored";
        Cache.Insert ( "item", item, null );
        DateTime.Now.AddSeconds ( 20 ), TimeSpan.Zero;
    }

    else
    {
        label1.Text + = "Item is accesses";
        DateTime item = ( DateTime) Cache [ "item" ];
        label1.Text + = "Time is: " + item.ToString();
        label1.Text + = <br/>";
    }

    label1.Text + = "<br/>";
}
```

Read ASP.NET Caching online: https://riptutorial.com/asp-net/topic/9148/asp-net-caching

Chapter 9: Data Binding

Examples

SQL Data Source

Controls that can be bound with data can make use of `SqlDataSource` controls. The `SqlDataSource` control not only allows you to retrieve data from a database, but also edit and sort the data.

Retrieving Data

Stored Procedure:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%$ ConnectionStrings:MyConnectionString %>"
    SelectCommand="sp_GetEmployees"
    SelectCommandType="StoredProcedure">
</asp:SqlDataSource>
```

SQL Query:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%$ ConnectionStrings:MyConnectionString %>"
    SelectCommand="SELECT
                        EmployeeID,
                        EmployeeFirstName,
                        EmployeeLastName
                    FROM
                        dbo.Employees">
</asp:SqlDataSource>
```

Parameters:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%$ ConnectionStrings:MyConnectionString %>"
    SelectCommand="SELECT
                        EmployeeID,
                        EmployeeFirstName,
                        EmployeeLastName
                    FROM
                        dbo.Employees
                    WHERE
                        DepartmentID = @DepartmentID;">
    <SelectParameters>
        <asp:ControlParameter ControlID="ddlDepartment"
            Name="DepartmentID"
            PropertyName="SelectedValue" />
    </SelectParameters>
```

```
</asp:SqlDataSource>
```

Be aware of the `CancelSelectOnNullParameter` option, that if set to true (default) will stop the data binding if any parameter is NULL

Basic Usage

GridView:

```
<asp:GridView ID="GridViewEmployees"
    runat="server"
    AutoGenerateColumns="false"
    DataSourceID="SqlDataSourceEmployees">
    <Columns>
        <asp:BoundField DataField="EmployeeID" HeaderText="Employee ID" />
        <asp:BoundField DataField="EmployeeFirstName" HeaderText="First Name" />
        <asp:BoundField DataField="EmployeeLastName" HeaderText="Last Name" />
    </Columns>
</asp:GridView>
```

Object Data Source

```
<asp:ObjectDataSource ID="ObjectDataSourceEmployees" runat="server"
    TypeName="MyPackage.MyDataAccessClass"
    DataObjectTypeName="MyPackage.Employee"
    SelectMethod="GetEmployees"
    UpdateMethod="SaveEmployee"
    InsertMethod="SaveEmployee">
</asp:ObjectDataSource>
```

In the code behind

The Data Access Class

```
public class MyDataAccess
{
    public static List<Employee> GetEmployees()
    {
        List<Employee> results = new List<Employee>()
        {
            new Employee(){ Id=1, Name="John Smith" },
            new Employee(){ Id=2, Name="Mary Jane" }
        };

        return results;
    }

    public static void SaveEmployee(Employee e)
    {
        // Persist Employee e to the DB/cache etc. here
    }
}
```

The Employee Class

```
public class Employee
{
    public Int32EmployeeId { get; set; }
    public string Name { get; set; }
}
```

Read Data Binding online: https://riptutorial.com/asp-net/topic/2245/data-binding

Chapter 10: Data List

Syntax

1. **ItemTemplate**:It potrays the content and layout of items within the list.This is mandatory Required
2. **AlternatingItemTemplate**:If mentioned, determines the content and layout of alternating items. If not mentioned, ItemTemplate is used.
3. **SeparatorTemplate** : If mentioned, is rendered between items (and alternating items). If not mentioned, a separator is not rendered.
4. **SelectedItemTemplate** : If mentioned, determines the content and layout of the selected item. If not mentioned, ItemTemplate (AlternatingItemTemplate) is used.
5. **EditItemTemplate** :If mentioned, determines the content and layout of the item being edited. If not mentioned, ItemTemplate (AlternatingItemTemplate, SelectedItemTemplate) is used.
6. **HeaderTemplate**:If mentioned, determines the content and layout of the list header. If not mentioned, the header is not rendered.
7. **FooterTemplate**:If mentioned, determines the content and layout of the list footer. If not mentioned, the footer is not rendered.

Examples

Data Binding in asp.net

Aspx

```
<asp:DataList runat="server" CssClass="sample" RepeatLayout="Flow" ID="dlsamplecontent"
RepeatDirection="Vertical" OnItemCommand="dlsamplecontent_ItemCommand">
       <ItemStyle CssClass="tdContainer" />
           <ItemTemplate>
             //you code
                    </ItemTemplate>
                </asp:DataList>
```

Aspx.cs

```
public void GetSamplingContentType()
    {
        try
        {
            ErrorLogger.gstrClientMethodName = this.GetType().FullName + "_" +
System.Reflection.MethodBase.GetCurrentMethod().Name + " : ";

         DataTable dt = new DataTable();
            dlsamplecontent.DataSource = dt;
            dlsamplecontent.DataBind();

        }
        catch (Exception ex)
        {
```

```
            ErrorLogger.ClientErrorLogger(ex);
        }
    }
```

Item Command and Retrieving Id using Command argument

```
protected void dlsamplecontent_ItemCommand(object source, DataListCommandEventArgs e)
    {

        try
        {
            int BlogId = Convert.ToInt32(e.CommandArgument.ToString());
            if (e.CommandName == "SampleName")
            {
               //your code

            }
        }
        catch (Exception ex)
        {
            ErrorLogger.ClientErrorLogger(ex);
        }
    }
```

Read Data List online: https://riptutorial.com/asp-net/topic/7041/data-list

Chapter 11: DayPilot Scheduler

Parameters

Parameter	Desc
DataStartField	specifies the data source column that contains event start (DateTime)
DataStartField	specifies the data source column that contains event start (DateTime)
DataEndField	specifies the data source column that contains event end (DateTime)
DataTextField	specifies the data soruce column that contains event text (string)
DataIdField	specifies the data source column that contains event id (string or integer)
DataResourceField	specifies the data soruce column that contains event resource foreign key (string)

Remarks

This is basics of DayPilot schedular which needs to be further explore.

Examples

Basic Info

DayPilot Scheduler widget displays a time line for multiple resources. Supports AJAX and HTML5. Automatic and manual localization. Full CSS styling support

Declaration

```
<%@ Register Assembly="DayPilot" Namespace="DayPilot.Web.Ui" TagPrefix="DayPilot" %>
 <DayPilot:DayPilotScheduler
 ID="DayPilotScheduler1"
 runat="server"

 DataStartField="eventstart"
 DataEndField="eventend"
 DataTextField="name"
 DataIdField="id"
 DataResourceField="resource_id"

 CellGroupBy="Month"
 Scale="Day"
```

```
 EventMoveHandling="CallBack"
 OnEventMove="DayPilotScheduler1_EventMove"  >

</DayPilot:DayPilotScheduler>
```

Read DayPilot Scheduler online: https://riptutorial.com/asp-net/topic/6027/daypilot-scheduler

Chapter 12: Directives

Examples

The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file. The basic syntax of Application directive is:

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

The attributes of the Application directive are:

Attributes	Description
Inherits	The name of the class from which to inherit.
Description	The text description of the application. Parsers and compilers ignore this.
Language	The language used in code blocks.

The Control Directive

The control directive is used with the user controls and appears in the user control (.ascx) files.

The basic syntax of Control directive is:

```
<%@ Control Language="C#"  EnableViewState="false" %>
```

The attributes of the Control directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables automatic association of events to handlers.
ClassName	The file name for the control.
Debug	The Boolean value that enables or disables compiling with debug symbols.
Description	The text description of the control page, ignored by compiler.
EnableViewState	The Boolean value that indicates whether view state is maintained across page requests.

Attributes	Description
Explicit	For VB language, tells the compiler to use option explicit mode.
Inherits	The class from which the control page inherits.
Language	The language for code and script.
Src	The filename for the code-behind class.
Strict	For VB language, tells the compiler to use the option strict mode.

The Implements Directive

The Implement directive indicates that the web page, master page or user control page must implement the specified .Net framework interface.

The basic syntax for implements directive is:

```
<%@ Implements  Interface="interface_name" %>
```

The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

```
<%@ MasterPage Language="C#"  AutoEventWireup="true"  CodeFile="SiteMater.master.cs"
Inherits="SiteMaster"  %>
```

The Import Directive

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

The basic syntax for import directive is:

```
<%@ namespace="System.Drawing" %>
```

The MasterType Directive

The MasterType directive assigns a class name to the Master property of a page, to make it strongly typed.

The basic syntax of MasterType directive is:

```
<%@ MasterType attribute="value"[attribute="value" ...]  %>
```

The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

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

The attributes of the Page directive are:

Attributes	Description
AutoEventWireup	The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page_Load.
Buffer	The Boolean value that enables or disables HTTP response buffering.
ClassName	The class name for the page.
ClientTarget	The browser for which the server controls should render content.
CodeFile	The name of the code behind file.
Debug	The Boolean value that enables or disables compilation with debug symbols.
Description	The text description of the page, ignored by the parser.
EnableSessionState	It enables, disables, or makes session state read-only.
EnableViewState	The Boolean value that enables or disables view state across page requests.
ErrorPage	URL for redirection if an unhandled page exception occurs.
Inherits	The name of the code behind or other class.
Language	The programming language for code.
Src	The file name of the code behind class.
Trace	It enables or disables tracing.
TraceMode	It indicates how trace messages are displayed, and sorted by time or category.
Transaction	It indicates if transactions are supported.

Attributes	Description
ValidateRequest	The Boolean value that indicates whether all input data is validated against a hardcoded list of values.

The OutputCache Directive

The OutputCache directive controls the output caching policies of a web page or a user control.

The basic syntax of OutputCache directive is:

```
<%@ OutputCache Duration="15" VaryByParam="None"  %>
```

Read Directives online: https://riptutorial.com/asp-net/topic/2255/directives

Chapter 13: Event Delegation

Syntax

1. `public delegate void ActionClick();`

   ```
   public event ActionClick OnResetClick;
   ```

Remarks

I haven't found any disadvantages in this approach but there are a few things which make this a little problematic.

1. You need to add an event handler for each and every event. If you do not add the event handlers in the OnInit event of the page, you might face some problems that on page post back, you will lose the event assignment (as ASP.NET is stateless, which is not the case with Windows controls).
2. In this approach, you need to respect the page life cycle events. Some times when you are working on the Designer, there might be a case when the event handler gets lost without your notice.
3. Even if you have not added the event handler, you will not get any errors or warnings. If you have multiple pages for performing the same action, there is no guarantee that all the method names will be same; the developer can choose their own method names, which reduces the maintainability of the code.

Examples

Delegation of Event from User Control to aspx

Normally, we opt this approach if we want complete encapsulation and don't want to make our methods public.

Ascx

```
<div style="width: 100%;">
  <asp:Button ID="btnAdd" runat="server"
    Text="Add" OnClick="btnAdd_Click"></asp:button>
  <asp:button id="btnEdit" runat="server"
    text="Edit" onclick="btnEdit_Click"> </asp:button>
  <asp:button id="btnDelete" runat="server"
    text="Delete" onclick="btnDelete_Click">    </asp:Button>
  <asp:button id="btnReset" runat="server"
    text="Reset" onclick="btnReset_Click"></asp:button>
</div>
```

Ascx.cs

```
public delegate void ActionClick();

public partial class EventDelegation : System.Web.UI.UserControl
{
    public event ActionClick OnAddClick;
    public event ActionClick OnDeleteClick;
    public event ActionClick OnEditClick;
    public event ActionClick OnResetClick;
    protected void btnAdd_Click(object sender, EventArgs e)
    {
        if(OnAddClick!= null)
        {
            OnAddClick();
        }
    }

    protected void btnEdit_Click(object sender, EventArgs e)
    {
        if (OnEditClick != null)
        {
            OnEditClick();
        }
    }

    protected void btnDelete_Click(object sender, EventArgs e)
    {
        if(OnDeleteClick!= null)
        {
            OnDeleteClick();
        }
    }

    protected void btnReset_Click(object sender, EventArgs e)
    {
        if(OnResetClick!= null)
        {
            OnResetClick();
        }
    }
}
```

The user control specifies some public events like `OnAddClick`, `OnEditClick`,etc., which declare a delegate. Anyone who wants to use these events needs to add the EventHandler to execute when the corresponding button click event occurs.

Aspx Design

```
<%@ Register src="Controls/EventDelegation.ascx"
       tagname="EventDelegation" tagprefix="uc1" %>

    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

    <html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title></title>
    </head>
    <body>
        <form id="form1" runat="server">
```

```
        <div>
            <uc1:Direct ID="Direct1" runat="server" />
        </div>
        </form>
    </body>
    </html>
```

Aspx.cs

```
public partial class EventDelegation : System.Web.UI.Page
{
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        ActionControl.OnAddClick += ActionControl_OnAddClick;
        ActionControl.OnDeleteClick += ActionControl_OnDeleteClick;
        ActionControl.OnEditClick += ActionControl_OnEditClick;
        ActionControl.OnResetClick += ActionControl_OnResetClick;
    }

    private void ActionControl_OnResetClick()
    {
        Response.Write("Reset done.");
    }

    private void ActionControl_OnEditClick()
    {
        Response.Write("Updated.");
    }

    private void ActionControl_OnDeleteClick()
    {
        Response.Write("Deleted.");
    }

    private void ActionControl_OnAddClick()
    {
        Response.Write("Added.");
    }
}
```

Read Event Delegation online: https://riptutorial.com/asp-net/topic/6927/event-delegation

Chapter 14: Event Handling

Syntax

- private void EventName (object sender, EventArgs e);

Parameters

Parameter	Details
object sender	sender refers to the object that invoked the event that fired the event handler. This is useful if you have many objects using the same event handler.
EventArgs e	EventArgs is something of a dummy base class. In and of itself it's more or less useless, but if you derive from it, you can add whatever data you need to pass to your event handlers.

Examples

Application and Session Events

The most important application events are:

Application_Start - It is raised when the application/website is started.

Application_End - It is raised when the application/website is stopped.

Similarly, the most used Session events are:

Session_Start - It is raised when a user first requests a page from the application.

Session_End - It is raised when the session ends.

Page and Control Events

Common page and control events are:

DataBinding - It is raised when a control binds to a data source.

Disposed - It is raised when the page or the control is released.

Error - It is a page event, occurs when an unhandled exception is thrown.

Init - It is raised when the page or the control is initialized.

Load - It is raised when the page or a control is loaded.

PreRender - It is raised when the page or the control is to be rendered.

Unload - It is raised when the page or control is unloaded from memory.

Default Events

The default event for the Page object is Load event. Similarly, every control has a default event. For example, default event for the button control is the Click event.

The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:

Control	Default Event
AdRotator	AdCreated
BulletedList	Click
Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
DataGrid	SelectedIndexChanged
DataList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu MenuItem	Click
RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

Example This example includes a simple page with a label control and a button control on it. As the page events such as Page_Load, Page_Init, Page_PreRender etc. take place, it sends a

message, which is displayed by the label control. When the button is clicked, the Button_Click event is raised and that also sends a message to be displayed on the label.

Create a new website and drag a label control and a button control on it from the control tool box. Using the properties window, set the IDs of the controls as .lblmessage. and .btnclick. respectively. Set the Text property of the Button control as 'Click'.

The markup file (.aspx):

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
   Inherits="eventdemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

   <head runat="server">
      <title>Untitled Page</title>
   </head>

   <body>
      <form id="form1" runat="server">
         <div>
            <asp:Label ID="lblmessage" runat="server" >

            </asp:Label>

            <br />
            <br />
            <br />

            <asp:Button ID="btnclick" runat="server" Text="Click" onclick="btnclick_Click" />
         </div>
      </form>
   </body>

</html>
```

Double click on the design view to move to the code behind file. The Page_Load event is automatically created without any code in it. Write down the following self-explanatory code lines:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;
```

```
namespace eventdemo {

   public partial class _Default : System.Web.UI.Page {

      protected void Page_Load(object sender, EventArgs e) {
         lblmessage.Text += "Page load event handled. <br />";

         if (Page.IsPostBack) {
            lblmessage.Text += "Page post back event handled.<br/>";
         }
      }

      protected void Page_Init(object sender, EventArgs e) {
         lblmessage.Text += "Page initialization event handled.<br/>";
      }

      protected void Page_PreRender(object sender, EventArgs e) {
         lblmessage.Text += "Page prerender event handled. <br/>";
      }

      protected void btnclick_Click(object sender, EventArgs e) {
         lblmessage.Text += "Button click event handled. <br/>";
      }
       }
    }
```

Execute the page. The label shows page load, page initialization and, the page pre-render events. Click the button to see effect:

Read Event Handling online: https://riptutorial.com/asp-net/topic/2347/event-handling

Chapter 15: Expressions

Examples

Value From App.Config

```
<asp:Literal runat="server" text="<%$ AppSettings:MyAppSettingName %>"/>
```

Evaluated Expression

```
<div>
  The time is now <%= DateTime.Now.ToString() %>
</div>
```

Code Block Within ASP Markup

```
<div>
    <form id="form1" runat="server">
        <%
            for (int i = 1; i <= 10; j++)
            {
                Response.Write(i) + " ";
            }
        %>
    </form>
<div>
```

Read Expressions online: https://riptutorial.com/asp-net/topic/6326/expressions

Chapter 16: Find Control by ID

Syntax

1. `control.FindControl("Id Of The Control To Be Found")`

Remarks

- `FindControl` is not recursive, it only searches through immediate children of the control
- There is an overload `FindControl(String, int)` which is not indented for public usage
- If nothing is found, `FindControl` returns `null`, so this is often a good idea to verify result for being not `null`

Examples

Accessing the TextBox Control in aspx Page

```
TextBox txt = (TextBox)FindControl(yourtxt_Id);
```

Find a control in a GridView, Repeater, ListView etc.

If the Control has rows.

```
TextBox tb = GridView1.Rows[i].FindControl("TextBox1") as TextBox;
```

Or if it has items.

```
TextBox tb = Repeater1.Items[i].FindControl("TextBox1") as TextBox;
```

Read Find Control by ID online: https://riptutorial.com/asp-net/topic/6894/find-control-by-id

Chapter 17: GridView

Examples

Data Binding

There are two ways you can bind a GridView. You can either manually do it by setting the `DataSource` property and calling `DataBind()`, or you can use a DataSourceControl such as a `SqlDataSource`.

Manual Binding

Create your GridView:

```
<asp:GridView ID="gvColors" runat="server"></asp:GridView>
```

First create or retrieve the source data for the GridView. Next, assign the data to the GridView's `DataSource` property. Finally, call `DataBind()`.

```
List<string> colors = new List<string>();
colors.Add("Red");
colors.Add("Green");
colors.Add("Blue");

gvColors.DataSource = colors;
gvColors.DataBind();
```

DataSourceControl

Create your DataSourceControl:

```
<asp:SqlDataSource ID="sdsColors"
    runat="server"
    ConnectionString="<%$ MyConnectionString %>"
    SelectCommand="SELECT Color_Name FROM Colors">
</asp:SqlDataSource>
```

Create your GridView and set the `DataSourceID` property:

```
<asp:GridView ID="gvColors"
    runat="server"
    DataSourceID="sdsColors">
</asp:GridView>
```

Columns

There are seven different column types that can be used within a GridView.

```
<asp:GridView ID="GridView1" runat="server">
    <Columns>
        ...
    </Columns>
</asp:GridView>
```

BoundField:

```
<asp:BoundField DataField="EmployeeID" HeaderText="Employee ID" />
```

ButtonField:

```
<asp:ButtonField ButtonType="Button" HeaderText="Select Employee" Text="Select"/>
```

CheckBoxField:

```
<asp:CheckBoxField DataField="IsActive" HeaderText="Is Active" />
```

CommandField:

```
<asp:CommandField ShowDeleteButton="true"
    ShowEditButton="true"
    ShowInsertButton="true"
    ShowSelectButton="true" />
```

HyperLinkField:

```
<asp:HyperLinkField HeaderText="Employee Profile"
    DataNavigateUrlFields="EmployeeID"
    DataNavigateUrlFormatString="EmployeeProfile.aspx?EmployeeID={0}" />
```

ImageField:

```
<asp:ImageField HeaderText="Photo"
    DataImageUrlField="EmployeeID"
    DataImageUrlFormatString="/images/{0}" />
```

TemplateField:

```
<asp:TemplateField>
    <HeaderTemplate>
        Name
    </HeaderTemplate>
    <ItemTemplate>
        <asp:Label ID="lblEmployeeName"
            runat="server"
            Text='<&# Eval("EmployeeName") %>'></asp:Label>
    </ItemTemplate>
</asp:TemplateField>
```

Strongly Typed GridView

Starting with Asp.net 4.5 web controls can take advantage from strongly-typed binding to get IntelliSense support and compiletime errors.

Create a class, which holds your model:

```
public class Album
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Artist { get; set; }
}
```

Define the GridView control on your page:

```
<asp:GridView ID="Grid" runat="server" AutoGenerateColumns="false"
ItemType="YourNamespace.Album">
    <Columns>
        <asp:TemplateField HeaderText="Id">
            <ItemTemplate>
                <asp:Label ID="lblName" runat="server" Text="<%# Item.Id %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Name">
            <ItemTemplate>
                <asp:Label ID="lblName" runat="server" Text="<%# Item.Name %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Artist">
            <ItemTemplate>
                <asp:Label ID="lblCity" runat="server" Text="<%# Item.Artist %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

Load the data and bind it:

```
var albumList = new List<Album>
{
    new Album {Id = 1, Artist = "Icing (a Cake cover band)", Name = "Toppings Vol. 1"},
    new Album {Id = 2, Artist = "Fleetwood PC", Name = "Best of Windows"},
    new Album {Id = 3, Artist = "this.Bandnames", Name = "TH_ (Pronounced \"Thunderscore\")"},
};

Grid.DataSource = albumList;
Grid.DataBind();
```

Handling command event

GridViews allow commands to be sent from a GridView row. This is useful for passing row-specific information into an event handler as command arguments.

To subscribe to a command event:

```
<asp:GridView ID="GridView1" ... OnRowCommand="GridView1_RowCommand">
```

Buttons are the most common way to raise commands. They also support a way to specify command arguments. In this example, the argument is an `ID` of the item that the row represents.

```
<TemplateField>
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" runat="server"
                        CommandName="SampleCmd"
                        CommandArgument='<%# Eval("ID") %>'>
        </asp:LinkButton>
    </ItemTemplate>
</TemplateField>
```

Alternatively, one can use a `CommandField` column template that provides the most common command controls.

Handling of the event in code behind:

```
protected void GridView1_RowCommand(object source, GridViewCommandEventArgs e)
{
    if (e.CommandName == "SampleCmd")
    {
        var id = e.CommandArgument;
    }
}
```

Note that the `CommandName` used in this example is arbitrary and is a choice of the developer. There is, however, a set of predefined names that the GridView itself recognizes. Corresponding events are raised when these commands are fired.

Command Name	Events Raised
Cancel	RowCancelingEdit
Delete	RowDeleting, RowDeleted
Edit	RowEditing
Page	PageIndexChanging, PageIndexChanged
Select	SelectedIndexChanging, SelectedIndexChanged
Sort	Sorting, Sorted
Update	RowUpdating, RowUpdated

Paging

ObjectDataSource

If using an ObjectDataSource, almost everything is handled for you already, just simply tell the GridView to `AllowPaging` and give it a `PageSize`.

```
<asp:GridView ID="gvColors"
    runat="server"
    DataSourceID="sdsColors"
    AllowPaging="True"
    PageSize="5">
</asp:GridView>

<asp:SqlDataSource ID="sdsColors"
    runat="server"
    ConnectionString="<%$ MyConnectionString %>"
    SelectCommand="SELECT Color_ID, Color_Name FROM Colors">
</asp:SqlDataSource>
```

Color_ID	Color_Name
1	Red
2	Blue
3	Green
4	Yellow
5	Purple
1 2 3	

Color_ID	Color_Name
6	Orange
7	Black
8	White
9	Gray
10	Brown
1 2 3	

Color_ID	Color_Name
11	Pink
12	Turquoise
13	Maroon
1 2 3	

Manual Binding

If binding manually, you must handle the `PageIndexChanging` event. Simply set the `DataSource` and `PageIndex` and re-bind the GridView.

```
<asp:GridView ID="gvColors"
    runat="server"
    AllowPaging="True"
    PageSize="5"
    OnPageIndexChanging="gvColors_PageIndexChanging">
</asp:GridView>
```

C#

```
protected void gvColors_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    gvColors.DataSource = // Method to retrieve DataSource
    gvColors.PageIndex = e.NewPageIndex;
    gvColors.DataBind();
}
```

VB.NET

```
Protected Sub gvColors_PageIndexChanging(sender As Object, e As GridViewPageEventArgs)
{
    gvColors.DataSource = // Method to retrieve DataSource
    gvColors.PageIndex = e.NewPageIndex
    gvColors.DataBind()
```

```
}
```

Update Gridview on row click

Gridviews are more useful if we can update the view as per our need. Consider a view with a lock/unlock feature in each row. It can be done like:

Add an update panel:

```
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional"> </asp:UpdatePanel>
```

Add a ContentTemplate and Trigger inside your UpdatePanel:

```
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
    <ContentTemplate>
    </ContentTemplate>

    <Triggers>
    </Triggers>
</asp:UpdatePanel>
```

Add your GridView inside ContentTemplate:

```
<ContentTemplate>
<asp:GridView ID="GridView1" runat="server">
          <Columns>
                <asp:TemplateField>
                    <ItemTemplate>
                        <asp:ImageButton ID="imgDownload" runat="server" OnClientClick="return
confirm('Are you sure want to Lock/Unlock ?');"
                            CommandName="togglelock"
                            CommandArgument='<%#Container.DataItemIndex%>'/>

                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>

</ContentTemplate>
```

Here we are giving our GridView1 one constant column, for lock button. Mind it, databind has not taken place till now.

Time for DataBind: (on PageLoad)

```
using (SqlConnection con= new SqlConnection(connectionString))
{
            SqlCommand sqlCommand = new SqlCommand(" ... ", con);
            SqlDataReader reader = sqlCommand.ExecuteReader();
            GridView1.DataSource = reader;
            GridView1.DataBind();
}
```

Lock/Unlock image will be different as per the value of a certain column in your GridView.

Consider a case where your table contains an attribute/column titled "Lock Status". Now you wish to (1) hide that column just after DataBind and just before page rendering and (2) Assign different images to each row on basis of that hidden column value i.e. if Lock Status for a row is 0, assign it "lock.jpg", if status is 1 assign it "unlock.jpg". To do this, we'll use `OnRowDataBound` option of GridView, it mingles with your GridView, just before rendering each row to the HTML page.

```
<ContentTemplate>
<asp:GridView ID="GridView1" runat="server" OnRowDataBound="GridView1_RowDataBound"> ...
```

In cs file

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
    {

        if (e.Row.RowType == DataControlRowType.DataRow)
        {
            e.Row.Cells[8].Visible = false; //hiding the desired column which is column number
8 in this case
            GridView1.HeaderRow.Cells[8].Visible = false; //hiding its header
            ImageButton imgDownload = (ImageButton)e.Row.FindControl("imgDownload");
            string lstate = ((CheckBox)e.Row.Cells[8].Controls[0]).Checked.ToString();
            if (lstate == "True")
            { imgDownload.ImageUrl = "images/lock.png"; }
            else
            {
                imgDownload.ImageUrl = "images/unlock.png";
            }
        }
    }
```

Now the GridView will be rendered as we want, now let us implement button click events on that Lock/Unlock image button. Understand, that to perform a specific operation on a specific row, a command has to be given to that row and GridView provides us with the same functionality named `OnRowCommand`.

```
<ContentTemplate>
<asp:GridView ID="GridView1" runat="server" OnRowDataBound="GridView1_RowDataBound"
OnRowCommand="GridView1_RowCommand">
...
</ContentTemplate>
```

It'll create a function in cs file which takes an `object sender` and `GridViewCommandEventArgs e` With `e.CommandArgument` we can get the index of the row which gave the command Point to be noted here is that, a row can have multiple buttons and the cs code needs to know which button from that row gave the command. So we'll use `CommandName`

```
<asp:ImageButton ID="imgDownload" runat="server" OnClientClick="return confirm('Are you sure
want to Lock/Unlock ?');"
                            CommandName="togglelock"
                            CommandArgument='<%#Container.DataItemIndex%>'/>
```

Now in the backend one can distinguish commands from different rows and different buttons.

```
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
    {
        if (e.CommandName == "togglelock")
        {
            using (SqlConnection con= new SqlConnection(connectionString))
            {
               int index = Convert.ToInt32(e.CommandArgument);
               SqlCommand sqlCommand = new SqlCommand(" ... ", con);
               SqlDataReader reader = sqlCommand.ExecuteReader();
               GridView1.DataSource = reader;
               GridView1.DataBind();
            }
        }
    }
```

Add `<asp:PostBackTrigger ControlID="GridView1"/>` to the `Trigger` and it will update the GridView once the DataBind is done.

Use `HorizontalAlign="Center"` to place the GridView at the center of the page.

Read GridView online: https://riptutorial.com/asp-net/topic/1680/gridview

Chapter 18: httpHandlers

Examples

Using an httpHandler (.ashx) to download a file from a specific location

Create a new httpHandler inside your ASP.NET project. Apply the following code (VB) to the handler file:

```
Public Class AttachmentDownload
    Implements System.Web.IHttpHandler

    Sub ProcessRequest(ByVal context As HttpContext) Implements IHttpHandler.ProcessRequest

        ' pass an ID through the query string to append a unique identifer to your
downloadable fileName

        Dim fileUniqueId As Integer = CInt(context.Request.QueryString("id"))

        ' file path could also be something like "C:\FolderName\FilesForUserToDownload

        Dim filePath As String = "\\ServerName\FolderName\FilesForUserToDownload"
        Dim fileName As String = "UserWillDownloadThisFile_" & fileUniqueId
        Dim fullFilePath = filePath & "\" & fileName
        Dim byteArray() As Byte = File.ReadAllBytes(fullFilePath)

        ' promt the user to download the file

        context.Response.Clear()
        context.Response.ContentType = "application/x-please-download-me" ' "application/x-
unknown"
        context.Response.AppendHeader("Content-Disposition", "attachment; filename=" &
fileName)
        context.Response.BinaryWrite(byteArray)
        context.Response.Flush()
        context.Response.Close()
        byteArray = Nothing

    End Sub

    ReadOnly Property IsReusable() As Boolean Implements IHttpHandler.IsReusable
        Get
            Return False
        End Get
    End Property

End Class
```

You can call the handler from code behind, or from a client side language. In this example I am using a javascript which will call the handler.

```
function openAttachmentDownloadHandler(fileId) {

    // the location of your handler, and query strings to be passed to it
```

```
    var url = "..\\_Handlers\\AttachmentDownload.ashx?";
    url = url + "id=" + fileId;

    // opening the handler will run its code, and it will close automatically
    // when it is finished.

    window.open(url);

}
```

Now attach that assign the javascript function to a button click event on a clickable element in your web form. For example:

```
<asp:LinkButton ID="lbtnDownloadFile" runat="server"
OnClientClick="openAttachmentDownloadHandler(20);">Download A File</asp:LinkButton>
```

Or you can call the javascript function from the code behind as well:

```
ScriptManager.RegisterStartupScript(Page,
                Page.GetType(),
                "openAttachmentDownloadHandler",
                "openAttachmentDownloadHandler(" & fileId & ");",
                True)
```

Now when you click your button the httpHandler will get your file to the browser and ask the user if they would like to download it.

Read httpHandlers online: https://riptutorial.com/asp-net/topic/3476/httphandlers

Chapter 19: Katana

Introduction

What Is Katana? Katana is a set of open source components for building and hosting OWIN-based web applications, maintained by the Microsoft Open Technologies Group.Katana provides an implementation of the OWIN specification, and is in fact used in an increasing number of ASP.NET project templates. Additionally, Katana provides a wide variety of ready-to-use middleware components, ready for use in an OWIN-based application.

Examples

Example

Basic KatanaConsole Application

```
namespace KatanaConsole
{
    // use an alias for the OWIN AppFunc:
    using AppFunc = Func<IDictionary<string, object>, Task>;

    class Program
    {
        static void Main(string[] args)
        {
            WebApp.Start<Startup>("http://localhost:8080");
            Console.WriteLine("Server Started; Press enter to Quit");
            Console.ReadLine();
        }
    }

    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            var middleware = new Func<AppFunc, AppFunc>(MyMiddleWare);
            app.Use(middleware);
        }

        public AppFunc MyMiddleWare(AppFunc next)
        {
            AppFunc appFunc = async (IDictionary<string, object> environment) =>
            {
                // Do something with the incoming request:
                var response = environment["owin.ResponseBody"] as Stream;
                using (var writer = new StreamWriter(response))
                {
                    await writer.WriteAsync("<h1>Hello from My First Middleware</h1>");
                }
                // Call the next Middleware in the chain:
                await next.Invoke(environment);
            };
            return appFunc;
```

```
            }
        }
    }
```

Read Katana online: https://riptutorial.com/asp-net/topic/8236/katana

Chapter 20: Middleware

Parameters

Parameter	Details
`IDictionary<string,object> environment`	This is the only collection in which OWIN communicates information during a call. All keys can be found at https://docs.asp.net/en/latest/fundamentals/owin.html#owin-keys

Remarks

The `AppFunc` type is just an alias for `Func<IDictionary<string, object>, Task>` type to shorten method signatures, much like `typedef` in C++.

Examples

Output the request path and the time it took to process it

```
//define a short alias to avoid chubby method signatures
using AppFunc = Func<IDictionary<string, object>, Task>;

class RequestTimeMiddleware
{
    private AppFunc _next;

    public RequestTimeMiddleware(AppFunc next)
    {
        _next = next;
    }

    public async Task Invoke(IDictionary<string, object> environment)
    {
        IOwinContext context = new OwinContext(environment);

        var path = context.Request.Path;
        var sw = Stopwatch.StartNew();
        //Queue up the next middleware in the pipeline
        await _next(environment);
        //When the request comes back, log the elapsed time
        Console.WriteLine($"Request for {path} processed in {sw.ElapsedMilliseconds}ms");
    }
}

public static class RequestTimeMiddlewareExtensions
{
    //Extension method as syntactic sugar, to get a meaningful way
    //in adding the middleware to the pipeline
    public static void UseRequestTimeMiddleware(this IAppBuilder app)
```

```
    {
        app.Use<RequestTimeMiddleware>();
    }
}


public class Startup
{
    public void Configuration(IAppBuilder app)
    {
        //add the Middleware as early as possible
        app.UseRequestTimeMiddleware();
        //Queue up every other module
        app.Use(async (environment, next) =>
        {
            await environment.Response.WriteAsync("Hello from the console world");
            await next();
        });
    }
}
```

Read Middleware online: https://riptutorial.com/asp-net/topic/6607/middleware

Chapter 21: Page Life Cycle

Examples

Life Cycle Events

Following are the page life cycle events:

PreInit - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overriding the OnPreInit method or creating a Page_PreInit handler.

Init - Init event initializes the control property and the control tree is built. This event can be handled by overriding the OnInit method or creating a Page_Init handler.

InitComplete - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.

LoadViewState - LoadViewState event allows loading view state information into the controls.

LoadPostData - During this phase, the contents of all the input fields are defined with the tag are processed.

PreLoad - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overriding the OnPreLoad method or creating a Page_PreLoad handler.

Load - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overriding the OnLoad method or creating a Page_Load handler.

LoadComplete - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overriding the OnLoadComplete method or creating a Page_LoadComplete handler

PreRender - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.

PreRenderComplete - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.

SaveStateComplete - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.

UnLoad - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and

references, such as database connections, are freed. This event can be handled by overriding the OnUnLoad method or creating a Page_UnLoad handler.

Code Example

```
using System;

namespace myProject
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        public string PageSteps = string.Empty;

        //Raised after the start stage is complete and before the initialization stage begins.
        protected void Page_PreInit(object sender, EventArgs e)
        {
            PageSteps += "1 - Page_PreInit<br>";

            //Access to page Controls not available in this step
            //Label1.Text = "Step 1";
        }

        //Raised after all controls have been initialized and any skin settings have been
applied.
        //The Init event of individual controls occurs before the Init event of the page.
        protected void Page_Init(object sender, EventArgs e)
        {
            PageSteps += "2 - Page_Init<br>";

            Label1.Text = "Step 2";
        }

        //Raised at the end of the page's initialization stage.
        //Only one operation takes place between the Init and InitComplete events: tracking of
view state changes is turned on.
        //View state tracking enables controls to persist any values that are programmatically
added to the ViewState collection.
        //Until view state tracking is turned on, any values added to view state are lost
across postbacks.
        //Controls typically turn on view state tracking immediately after they raise their
Init event.
        protected void Page_InitComplete(object sender, EventArgs e)
        {
            PageSteps += "3 - Page_InitComplete<br>";

            Label1.Text = "Step 3";
        }

        //Raised after the page loads view state for itself and all controls, and after it
processes postback data that is included with the Request instance.
        protected override void OnPreLoad(EventArgs e)
        {
            PageSteps += "4 - OnPreLoad<br>";

            Label1.Text = "Step 4";
        }

        //The Page object calls the OnLoad method on the Page object, and then recursively
does the same for each child control until the page and all controls are loaded.
```

```
        //The Load event of individual controls occurs after the Load event of the page.
        protected void Page_Load(object sender, EventArgs e)
        {
            PageSteps += "5 - Page_Load<br>";

            Label1.Text = "Step 5";
        }

        //Use these events to handle specific control events, such as a Button control's Click
event or a TextBox control's TextChanged event.
        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            //Step only visible on PostBack
            PageSteps += "6 - btnSubmit_Click<br>";

            Label1.Text = "Step 6";
        }

        //Raised at the end of the event-handling stage.
        protected void Page_LoadComplete(object sender, EventArgs e)
        {
            PageSteps += "7 - Page_LoadComplete<br>";

            Label1.Text = "Step 7";
        }

        //Raised after the Page object has created all controls that are required in order to
render the page, including child controls of composite controls.
        //(To do this, the Page object calls EnsureChildControls for each control and for the
page.)
        protected override void OnPreRender(EventArgs e)
        {
            PageSteps += "8 - OnPreRender<br>";

            Label1.Text = "Step 8";
        }

        //Raised after each data bound control whose DataSourceID property is set calls its
DataBind method.
        protected override void OnPreRenderComplete(EventArgs e)
        {
            PageSteps += "9 - OnPreRenderComplete<br>";

            Label1.Text = "Step 9";
        }


        //Raised after view state and control state have been saved for the page and for all
controls.
        //Any changes to the page or controls at this point affect rendering, but the changes
will not be retrieved on the next postback.
        protected override void OnSaveStateComplete(EventArgs e)
        {
            PageSteps += "10 - OnSaveStateComplete<br><hr><br>";

            Label1.Text = "Step 10";
        }

        // Render
        //This is not an event; instead, at this stage of processing, the Page object calls
this method on each control.
```

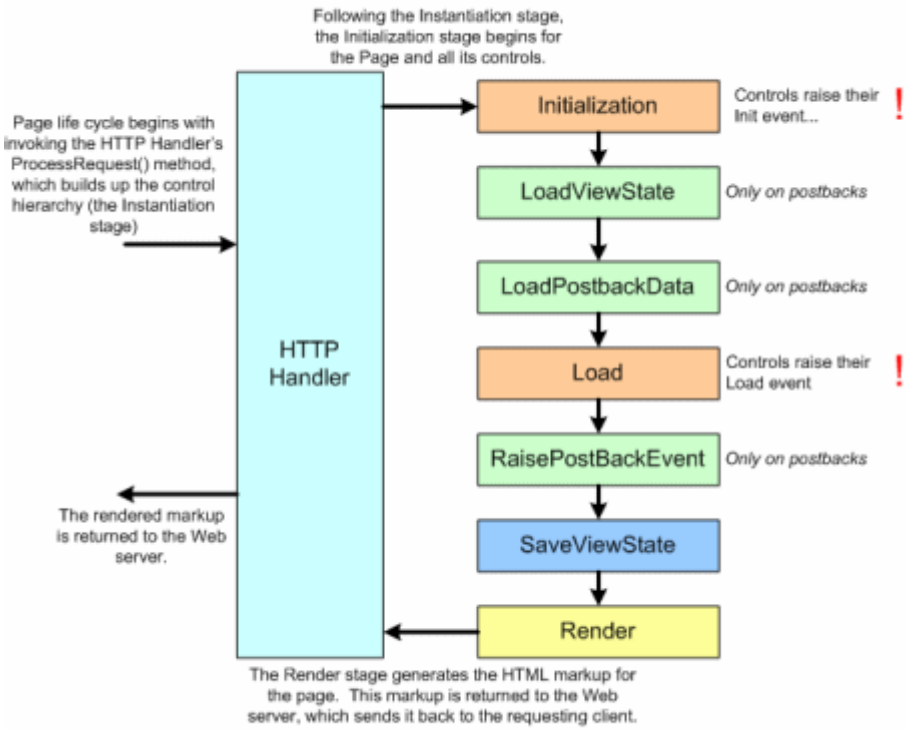
```
        //All ASP.NET Web server controls have a Render method that writes out the control's
markup to send to the browser.

        //Raised for each control and then for the page.
        //Controls use this event to do final cleanup for specific controls, such as closing
control-specific database connections
        protected void Page_UnLoad(object sender, EventArgs e)
        {
            //This last PageSteps addition will not be visible on the page
            PageSteps += "11 - Page_UnLoad<br>";

            //Access to page Controls not available in this step
            //Label1.Text = "Step 11";
        }
    }
}
```

Add the following code to the .aspx page to visualize the Steps in the Life Cycle.

```
<b>Page Life Cycle Visualization:</b>
<br />
<%= PageSteps %>
```



More information

- https://msdn.microsoft.com/en-us/library/ms178472.aspx
- https://www.tutorialspoint.com/asp.net/asp.net_life_cycle.htm
- http://www.c-sharpcorner.com/UploadFile/8911c4/page-life-cycle-with-examples-in-Asp-Net/
- https://www.codeproject.com/Articles/667308/ASP-NET-Page-Life-Cycle-Events

Read Page Life Cycle online: https://riptutorial.com/asp-net/topic/4948/page-life-cycle

Chapter 22: Page Methods

Parameters

Parameter	Detail
limit	The parameter of the C# method. You supply the argument via the Page Method.
onSuccess	The JavaScript function that is executed when the Page Method call is successful.
onError	The JavaScript function that is executed when there is an error in the Page Method call.

Remarks

More than one parameter

In the example the C# function just request one parameter, if you need to pass more than one you can do it, just put them in order in your JS call and you are good to go. Ej.

```
//C#
public static int SumValues(int num1, int num2, int num3, ..., int numN)

//JS
PageMethods.SumValues(num1, num2, num3, ..., numN, onSuccess, onError);
```

Return value

In the `onSuccess` function the result is going to be the C# function's return value. In the `onError` function the result is going to be the error.

Examples

How to call it

Just add the `using` at the beginning and the `[WebMethod]` decorator to the `static` method to be called in the aspx page:

```
using System.Web.Services;
```

```
public partial class MyPage : System.Web.UI.Page
{
    [WebMethod]
    public static int GetRandomNumberLessThan(int limit)
    {
        var r = new Random();
        return r.Next(limit);
    }
}
```

In your .aspx file add a asp:ScriptManager enabling Page Methods:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true">
</asp:ScriptManager>
```

Then you can call it from JS like this:

```
var limit= 42 // your parameter value
PageMethods.GetRandomNumberLessThan(limit, onSuccess, onError);
function onSuccess(result) {
    var randomNumber = result;
    // use randomNumber...
}
function onError(result) {
    alert('Error: ' + result);
}
```

Read Page Methods online: https://riptutorial.com/asp-net/topic/1411/page-methods

Chapter 23: Repeater

Examples

Basic usage

This example creates a simple 1-column repeater that displays a list of numbers, one per repeater item.

Markup:

```
<asp:Repeater ID="Repeater1" runat="server">
    <ItemTemplate>
        <%# Container.DataItem.ToString() %>
    </ItemTemplate>
</Repeater>
```

Code behind:

```
protected void Page_Load(object sender, EventArgs e)
{
    List<int> numbers = new List<int>{1, 2, 3, 4, 5};
    Repeater1.DataSource = numbers;
    Repeater1.DataBind();
}
```

Read Repeater online: https://riptutorial.com/asp-net/topic/2635/repeater

Chapter 24: ScriptManager

Introduction

ScriptManager control registers the script for the Microsoft AJAX Library with the page. This enables client script support features such as partial-page rendering and Web-service calls.

Syntax

1. <asp:ScriptManager ID="smPop" runat="server"></asp:ScriptManager>
2. ScriptManager.RegisterStartupScript(Control,Type,String,String,Boolean);

Examples

Working with ScriptManager

You must use a ScriptManager control on a page to enable the following features of ASP.NET AJAX:

1. Client-script functionality of the Microsoft AJAX Library, and any custom script that you want to send to the browser.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Page.ClientScript.RegisterStartupScript(
        this.GetType(),"myscript","alert('hello world!');");
}
```

2. Partial-page rendering, which enables regions on the page to be independently refreshed without a postback. The ASP.NET AJAX UpdatePanel, UpdateProgress, and Timer controls require a ScriptManager control to support partial-page rendering.

3. JavaScript proxy classes for Web services, which enable you to use client script to access Web services by exposing Web services as strongly typed objects.

```
[WebMethod]
public int Add(int a, int b) { return a + b; }

function CallAdd()
{
    // method will return immediately
    // processing done asynchronously
    WebService.Add(0,6, OnMethodSucceeded, OnMethodFailed);
}
```

4. JavaScript classes to access ASP.NET authentication and profile application services.

```
Sys.Services.AuthenticationService.login
Sys.Services.AuthenticationService.logout

<script type="text/javascript">
    function MyMethod(username, password)
    {
        Sys.Services.AuthenticationService.login(username,
            password,false,null,null,null,null,"User Context");
    }
</script>
```

See more at https://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx

Read ScriptManager online: https://riptutorial.com/asp-net/topic/10077/scriptmanager

Chapter 25: Session Managment

Examples

Advantage and Disadvantage of Session State, types of session

```
 The advantages of using Session State are

    1)Better security
    2)Reduced bandwidth

The disadvantages of using Session state are

    1)More resource consumption of server.
    2)Extra code/care if a Web farm is used(we will discuss this shortly)


   **Session State Modes**

   1) InProc mode, which stores session state in memory on the Web server. This is the
default.

   2) StateServer mode, which stores session state in a separate process called the ASP.NET
state service. This ensures that session state is preserved if the Web application is
restarted and also makes session state available to multiple Web servers in a Web farm.

   3) SQLServer mode stores session state in a SQL Server database. This ensures that session
state is preserved if the Web application is restarted and also makes session state available
to multiple Web servers in a Web farm.

  4) Custom mode, which enables you to specify a custom storage provider.

    Off mode, which disables session state.
```

Read Session Managment online: https://riptutorial.com/asp-net/topic/4180/session-managment

Chapter 26: Session State

Syntax

- Session["Session_Key"] = Obj_Value;

Remarks

HTTP is stateless. ASP.NET session state is a framework that facilitates maintaining state between HTTP page requests.

Session differs from the class level variables in its ability to remain available across post-backs and different pages. For instance, a session variable created in Page1.aspx will be available if the user is redirected to Page2.aspx afterwards, within the same application.

Also, in contrast to static variables declared at the page level, the session variables are independent for different users. Meaning, changing the value of one user's session variable will not affect the value of the same variable for other users.

While `ViewState` can be used to store user's data temporarily, it doesn't allow saving data across multiple pages. In addition, the `viewstate` is part of the page and is sent to the client. As a result, any critical information related to the user cannot be saved in the `ViewState`, and that's where Session variables become useful.

Examples

Using the Session object to store values

The `System.Web.SessionState.HttpSessionState` object provides a way to persist values between HTTP requests. In the example below, a user's preference for warnings is being saved in the session. Later on, while serving another request to the user, the application can read this preference from session and hide the warnings.

```
public partial class Default : System.Web.UI.Page
{
    public void LoadPreferences(object sender, EventArgs args)
    {
        // ...
        // ... A DB operation that loads the user's preferences.
        // ...

        // Store a value with the key showWarnings
        HttpContext.Current.Session["showWarnings"] = false;
    }

    public void button2Clicked(object sender, EventArgs args)
    {
        // While processing another request, access this value.
```

```
        bool showWarnings = (bool)HttpContext.Current.Session["showWarnings"];
        lblWarnings.Visible = false;
    }
}
```

Note that the session variables are not common for all users (just like cookies), and they are persisted across multiple post-backs.

The session works by setting a cookie that contains an identifier for the users session. By default this identifier is stored in the web-server memory, along with the values stored against it.

Here is a screenshot of the cookie set in the user's browser to keep track of the session:

Name	Value
ASP.NET_SessionId	3235CC720E020D5F045

Using a SQL Session Store

If you find that you have multiple servers that need to share session state, storing it in the ASP.NET process memory will not work. For example you may deploy into a web-farm environment with a load balancer that distributes requests in a round-robin fashion. In this environment a single user's requests could be served by multiple servers.

In the web.config file you can configure a SQL server session store.

```
<configuration>
  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="Data Source=localhost;Integrated Security=SSPI"
      cookieless="true"
      timeout="30" />
  </system.web>
</configuration>
```

To create the sql schema use the aspnet_regsql tool. [SampleSqlServerName] is the hostname of the SQL server. -ssadd tells the tool to create the session state database. -sstype p tells the tool to create a new database with the default name ASPState.

```
aspnet_regsql.exe -S [SampleSqlServerName] -U [Username] -P [Password] -ssadd -sstype p
```

Using an Amazon DynamoDB Session Store

If you don't want to use SQL server you can use Amazon's hosted Dynamo DB nosql database as a session store.

You'll need the AWS SDK. To install this from the Visual Studio nuget package manager console use the following command

```
Install-Package AWSSDK
```

You can then configure your sessionState provider to use a custom provider. You must specify the region and credentials, either a profile or an IAM access and secret key combination. By default this will create a table named ASP.NET_SessionState.

```
<configuration>
  <system.web>
    <sessionState
      timeout="20"
      mode="Custom"
      customProvider="DynamoDBSessionStoreProvider">
      <providers>
        <add name="DynamoDBSessionStoreProvider"
             type="Amazon.SessionProvider.DynamoDBSessionStateStore"
             AWSProfileName="[PROFILE]"
             Region="[REGION]"
             CreateIfNotExist="true"
             />
      </providers>
    </sessionState>
  </system.web>
</configuration>
```

Read Session State online: https://riptutorial.com/asp-net/topic/3864/session-state

Chapter 27: UpdatePanel

Introduction

This topic describes how to add partial-page update support to a Web page by using two Microsoft Ajax server controls: the ScriptManager control and the UpdatePanel control. These controls remove the requirement to refresh the whole page with each postback, which improves the user experience.

Syntax

- <asp:UpdatePanel ID="UpdatePanel1" runat="server">
  </asp:UpdatePanel>

Remarks

A ScriptManager must be added to page to make the UpdatePanel to work.

Examples

Update Panel Example

Step 1: Add ScriptManager to your page

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
```

Step 2: Add UpdatePanel to your page just after ScriptManager.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate></ContentTemplate>
        </asp:UpdatePanel>
```

Step 3: After adding content to your UpdatePanels Content Template your aspx page should look something like this:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
    <style type="text/css">
    #UpdatePanel1 {
      width:300px; height:100px;
```

```
     }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div style="padding-top: 10px">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <fieldset>
                <legend>UpdatePanel</legend>
                <asp:Label ID="Label1" runat="server" Text="Panel created."></asp:Label><br />
                <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Button"
/>
                </fieldset>
            </ContentTemplate>
        </asp:UpdatePanel>
        <br />
        </div>

    </div>
    </form>
</body>
</html>
```

Step 4: Add this part to your C# page:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "Refreshed at " +
        DateTime.Now.ToString();
}
```

Step 5: Now run your application.

Expected Result:

The panel content changes every time that you click the button, but the whole page is not refreshed. By default, the ChildrenAsTriggers property of an UpdatePanel control is true. When this property is set to true, controls inside the panel participate in partial-page updates when any control in the panel causes a postback.

Read UpdatePanel online: https://riptutorial.com/asp-net/topic/10075/updatepanel

Chapter 28: View State

Introduction

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

Syntax

- ViewState["NameofViewstate"] = "Value";

Examples

Example

ASPX

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="_Default"
%>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>ViewState</title>
    </head>
    <body>
        <form id="form1" runat="server">
            <asp:TextBox runat="server" id="NameField" />
            <asp:Button runat="server" id="SubmitForm" onclick="SubmitForm_Click" text="Submit
& set name" />
            <asp:Button runat="server" id="RefreshPage" text="Just submit" />
            <br /><br />
            Name retrieved from ViewState: <asp:Label runat="server" id="NameLabel" />
        </form>
    </body>
    </html>
```

Code behind

```
using System;
using System.Data;
using System.Web;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if(ViewState["NameOfUser"] != null)
```

```
            NameLabel.Text = ViewState["NameOfUser"].ToString();
        else
            NameLabel.Text = "Not set yet...";
    }

    protected void SubmitForm_Click(object sender, EventArgs e)
    {
        ViewState["NameOfUser"] = NameField.Text;
        NameLabel.Text = NameField.Text;
    }
}
```

Read View State online: https://riptutorial.com/asp-net/topic/8234/view-state

Chapter 29: web.config > system.webServer/httpErrors & system.web/customErrors sections

Introduction

CustomErrors are a legacy (backwards compatable) element, used by Visual Studio Development Server (aka. VSDS or Cassini).

httpErrors are the new element which is only used by IIS7.

Examples

What is the difference between customErrors and httpErrors?

Both are used to define error handling for a website, but different software refers to different config elements.

customErrors are a legacy (backwards compatable) element, used by Visual Studio Development Server (aka. VSDS or Cassini).

httpErrors are the new element which is only used by IIS7.

This highlights the possible problem when developing ASP.NET websites while using VSDS instead of the local IIS.

Also, refer to this post by myself about how to handle error messages with IIS7, if you wish to have full control of the error output.

Summary:

1. Developing in VSDS - use customErrors
2. Publishing the site to IIS6 - use customErrors
3. Publishing the site to IIS7 - use httpErrors.
4. and if you develop with VSDS but publish to IIS7, then i guess u'll need both.

Read web.config > system.webServer/httpErrors & system.web/customErrors sections online: https://riptutorial.com/asp-net/topic/10103/web-config---system-webserver-httperrors---system-web-customerrors-sections

Chapter 30: WebForms

Syntax

- <asp:TextBox runat="server" ID="" TextMode="" Text="" />
- <asp:Repeater runat="server" ID="" OnItemDataBound="">
  `<HeaderTemplate></HeaderTemplate>`
  `<ItemTemplate></ItemTemplate>`
  `<FooterTemplate></FooterTemplate>`
  `</asp:Repeater>`

Remarks

All ASP.Net WebForm controls require `runat="server"` in order to communicate with the CodeBehind.

Examples

Using a Repeater to create a HTML Table

When the Repeater is Bound, for each item in the data, a new table row will be added.

```
<asp:Repeater ID="repeaterID" runat="server" OnItemDataBound="repeaterID_ItemDataBound">
    <HeaderTemplate>
        <table>
            <thead>
                <tr>
                    <th style="width: 10%">Column 1 Header</th>
                    <th style="width: 30%">Column 2 Header</th>
                    <th style="width: 30%">Column 3 Header</th>
                    <th style="width: 30%">Column 4 Header</th>
                </tr>
            </thead>
    </HeaderTemplate>
    <ItemTemplate>
        <tr runat="server" id="rowID">
            <td>
                <asp:Label runat="server" ID="mylabel">You can add ASP labels if you
want</asp:Label>
            </td>
            <td>
                <label>Or you can add HTML labels.</label>
            </td>
            <td>
                You can also just type plain text like this.
            </td>
            <td>
                <button type="button">You can even add a button to the table if you
want!</button>
            </td>
        </tr>
    </ItemTemplate>
```

```
    <FooterTemplate>
        </table>
    </FooterTemplate>
</asp:Repeater>
```

The `ItemDataBound` method is optional, yet useful for formatting or populating more complicated data. In this example, the method is used to dynamically give each `<tr>` a unique ID. This ID can then be use in JavaScript to access or modify a specific row. Note, the `tr` will not keep its dynamic ID value on PostBack. The text of each row's `<asp:Label>` was also set in this method.

```
protected void repeaterID_ItemDataBound(object sender, RepeaterItemEventArgs e)
{
    if (e.Item.ItemType == ListItemType.Item || e.Item.ItemType ==
ListItemType.AlternatingItem)
    {
        MyItem item = (MyItem)e.Item.DataItem;

        var row = e.Item.FindControl("rowID");
        row.ClientIDMode = ClientIDMode.Static;
        row.ID = "rowID" + item.ID;

        Label mylabel = (Label)e.Item.FindControl("mylabel");
        mylabel.Text = "The item ID is: " + item.ID;
    }
}
```

If you plan on doing a lot of communication with the CodeBehind, you might want to consider using GridView. Repeaters, however, in general have less overhead than GridView, and with basic ID manipulation, can perform the same functions as GridView.

Grouping in ListView

`asp:ListView` introduced in ASP.NET WebForms framework 3.5 is the most flexible of all DataPresentation Controls in the framework. An example of Grouping using ListView (which will come handy as an image gallery)

Objective: To display three images in a row using `asp:ListView`

Markup

```
<asp:ListView ID="SportsImageList" runat="server"
    GroupItemCount="3">
    <LayoutTemplate>
        <span class="images-list">
            <ul id="groupPlaceholder" runat="server"></ul>
        </span>
    </LayoutTemplate>
    <GroupTemplate>
        <ul>
            <li id="itemPlaceholder" runat="server"></li>
        </ul>
    </GroupTemplate>
    <ItemTemplate>
        <li>
```

```
            <img src='<%# Container.DataItem %>' />
        </li>
    </ItemTemplate>
</asp:ListView>
```

Code Behind

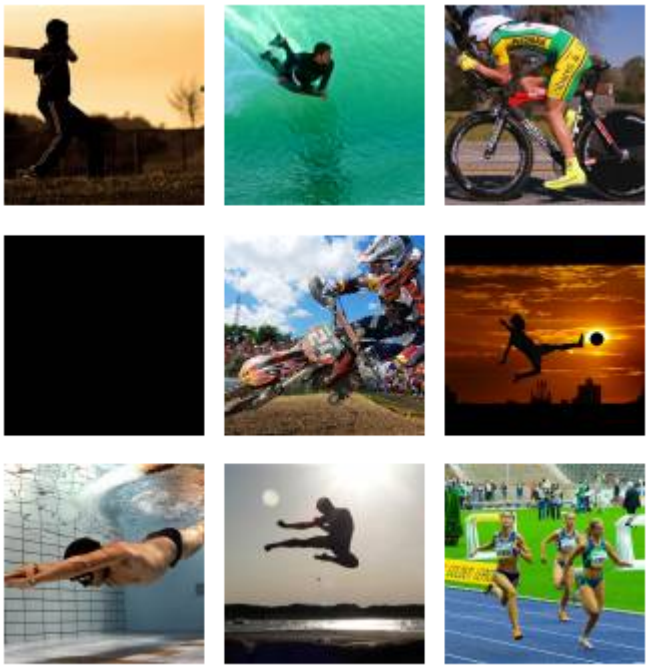
```
protected void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        SportsImageList.DataSource = GetImages();
        SportsImageList.DataBind();
    }
}

private static IEnumerable<string> GetImages()
{
    var images = Enumerable.Range(1, 9) //get numbers 1 to 9
        .Select(i =>
            string.Format("http://lorempixel.com/100/100/sports/{0}/", i)
        ); //convert the numbers to string
    return images;
}
```

CSS

```
.images-list ul{
    clear: both;
    list-style-type: none;
}
.images-list ul li{
    float: left;
    padding: 5px;
}
```

Rendered Output



Example

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

    Sub SubmitBtn_Click(sender As Object, e As EventArgs)
        Label1.Text = "Text1.Text = " & Text1.Text
    End Sub

</script>
```

```
<h3><font face="Verdana">TextBox Sample</font></h3>

<form runat="server">

  <asp:TextBox id="Text1" Text="Copy this text to the label" Width="200px" runat="server"/>

  <asp:Button OnClick="SubmitBtn_Click" Text="Copy Text to Label" Runat="server"/>

  <p>

  <asp:Label id="Label1" Text="Label1" runat="server"/>

</form>
```

Hyperlink

The HyperLink control is used to navigate from the client to another page.

```
<html>

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


   Sub Page_Load(sender As Object, e As EventArgs)
```

```
      ' Set hyperlink to "~", which indicates application root.
      HyperLink1.NavigateUrl = "~"
   End Sub

</script>

<body>

    <h3><font face="Verdana">Simple asp:hyperlink Sample</font></h3>

    <form runat=server>

        <p>

        <asp:hyperlink id=HyperLink1 runat="server">
            Go To QuickStart
        </asp:hyperlink>

    </form>

</body>

</html>
```

Read WebForms online: https://riptutorial.com/asp-net/topic/5394/webforms

Chapter 31: WebService without Visual Studio

Introduction

A very basic ASP.Net example of the bare minimum of code to create a WebService.

Remarks

In a separate StackOverflow Documentation post, we'll look at consuming this Calculator WebService.

Examples

Calculator WebService

```
<%@ WebService Language="C#" Class="Util" %>
using System;
using System.Web.Services;

public class Util: WebService
{
    [WebMethod]
    public int CalculatorAdd(int operandA, int operandB)
    {
        return operandA + operandB;
    }

    [WebMethod]
    public int CalculatorSubtract(int operandA, int operandB)
    {
        return operandA - operandB;
    }

    [WebMethod]
    public long CalculatorMultiply(int operandA, int operandB)
    {
        return operandA * operandB;
    }

    [WebMethod]
    public long CalculatorDivide(int operandNumerator, int operandDenominator)
    {
        if (operandDenominator == 0)
            return System.Int64.MaxValue;    // Should really do better error handling overall
& return an error
        else
            return operandNumerator / operandDenominator;
    }
}
```

Read WebService without Visual Studio online: https://riptutorial.com/asp-net/topic/8859/webservice-without-visual-studio

Credits

S. No	Chapters	Contributors
1	Getting started with ASP.NET	Ahmed Abdelhameed, Aristos, Community, demonplus, Dillie-O, Josh E, khawarPK, Marco, Matt, Muhammad Awais, Satinder singh, wintersolider
2	Asp Web Forms Identity	tatigo
3	ASP.NET - Basic Controls	khawarPK
4	ASP.NET - Managing State	khawarPK
5	ASP.NET - User Controls	Tetsuya Yamamoto
6	ASP.NET - Validators	khawarPK
7	Asp.net Ajax Controls	Saurabh Srivastava
8	ASP.NET Caching	tatigo
9	Data Binding	j.f., Ryan
10	Data List	Webruster
11	DayPilot Scheduler	Abdul
12	Directives	khawarPK, Tot Zam
13	Event Delegation	Webruster
14	Event Handling	khawarPK, Tot Zam
15	Expressions	Ryan
16	Find Control by ID	Andrei, VDWWD, Webruster
17	GridView	Andrei, Asif.Ali, j.f., Marco, Ritwik
18	httpHandlers	Taylor Brown
19	Katana	jignesh
20	Middleware	Marco
21	Page Life Cycle	Abdul, mbenegas, Srikar, VDWWD

22	Page Methods	Enrique Zavaleta, wazz, XIII
23	Repeater	Andrei
24	ScriptManager	Naveen Gogineni
25	Session Managment	Jasmin Solanki
26	Session State	Luke Ryan, Naveen Gogineni, Nisarg Shah
27	UpdatePanel	Naveen Gogineni
28	View State	jignesh
29	web.config > system.webServer/httpErrors & system.web/customErrors sections	Naveen Gogineni
30	WebForms	Big Fan, jignesh, naveen, Tot Zam
31	WebService without Visual Studio	George 2.0 Hope