

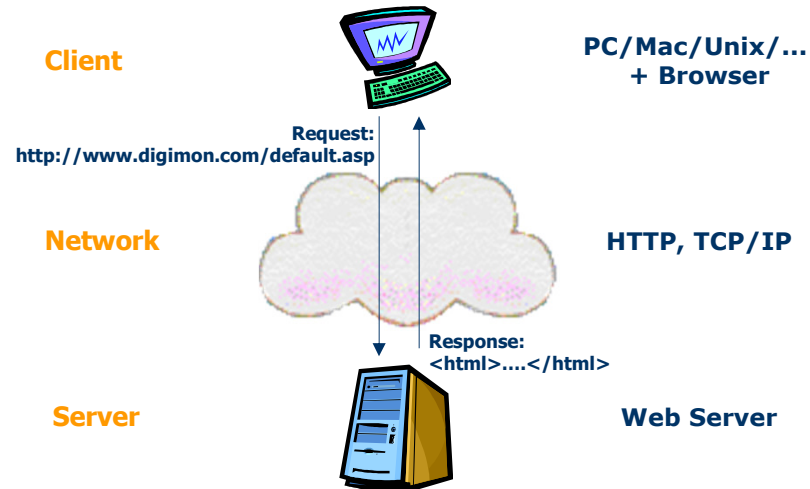
# Introduction to ASP.NET and Web Forms

This material is based on the original slides of Dr. Mark Sapossnek, Computer Science Department, Boston University, Mosh Teitelbaum, evoch, LLC, and Joe Hummel, Lake Forest College

## Outline

- ◆ **Background**
- ◆ ASP.NET Overview
- ◆ Programming Model
- ◆ Programming Basics
- ◆ Server Controls
- ◆ Data Binding
- ◆ Conclusion

## Background Web Architecture



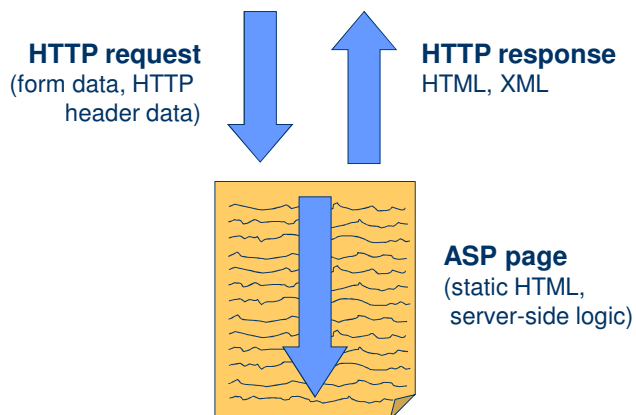
## Background Web Development Technologies

- ◆ Client-side technologies
  - XHTML, CSS, DOM, JavaScript
- ◆ Server-side technologies
  - ASP (Active Server Pages)
  - ASP.NET is the next generation of ASP

## Background What is ASP?

- ◆ Server-side programming technology
- ◆ Consists of static HTML interspersed with script
- ◆ ASP intrinsic objects (**Request, Response, Server, Application, Session**) provide services
- ◆ Commonly uses ADO to interact with databases
- ◆ Application and session variables
- ◆ Application and session begin/end events
- ◆ ASP manages threads, database connections.

## Background What is ASP?



## Background

### Example: HelloWorld.asp



```
<html>
<head><title>HelloWorld.asp</title></head>
<body>
<form method="post">
<input type="submit" id=button1 name=button1
value="Push Me" />
<%
if (Request.Form("button1") <> "") then
  Response.write "<p>Hello, the time is " & Now()
end if
%>
</form>
</body>
</html>
```

## Background

### ASP Challenges

- ◆ Coding overhead (too much code)
  - Everything requires writing code!
- ◆ Code readability (too complex; code and UI intermingled)
- ◆ Maintaining page state [After submit button is clicked, if we click the back button, we expect to maintain scroll position, maintain which control had focus, and restore focus, or allow server code to focus a new control ] requires more code
- ◆ Session state scalability and availability
- ◆ Limited support for caching, tracing, debugging, etc.
- ◆ Performance and safety limitations of script

## Outline

- ◆ Background
- ◆ **ASP.NET Overview**
- ◆ Programming Model
- ◆ Programming Basics
- ◆ Server Controls
- ◆ Data Binding
- ◆ Conclusion

## ASP.NET Overview

- ◆ ASP.NET provides services to allow the creation, deployment, and execution of Web Applications and Web Services
- ◆ Like ASP, ASP.NET is a server-side technology
- ◆ Web Applications are built using Web Forms
- ◆ Web Forms are designed to make building web-based applications as easy as building Visual Basic applications

## ASP.NET Overview

### Goals

- ◆ Keep the good parts of ASP and improve the rest
- ◆ Simplify: less code, easier to create and maintain
- ◆ Multiple, compiled languages
- ◆ Fast
- ◆ Scalable
- ◆ Manageable
- ◆ Available
- ◆ Customizable and extensible
- ◆ Secure
- ◆ Tool support

## ASP.NET Overview

### Key Features

- ◆ Web Forms
- ◆ Web Services
- ◆ Built on .NET Framework
- ◆ Simple programming model
- ◆ Maintains page state
- ◆ Multibrowser support
- ◆ XCOPY deployment
- ◆ XML configuration
- ◆ Complete object model
- ◆ Session management
- ◆ Caching
- ◆ Debugging
- ◆ Extensibility
- ◆ Separation of code and UI
- ◆ Security
- ◆ Simplified form validation
- ◆ Cookieless sessions

# ASP.NET Overview

## Example: HelloWorld.aspx



```
<%@ Page language="c#" %>
<html>
<head></head>
<script runat="server">
public void B_Click (object sender, System.EventArgs e) {
    Label1.Text = "Hello, the time is " + DateTime.Now;
}
</script>
<body>
    <form method="post" runat="server">
        <asp:Button onclick="B_Click" Text="Push Me"
            runat="server" /> <br>
        <asp:Label id=Label1 runat="server" />
    </form>
</body>
</html>
```

# ASP.NET Overview

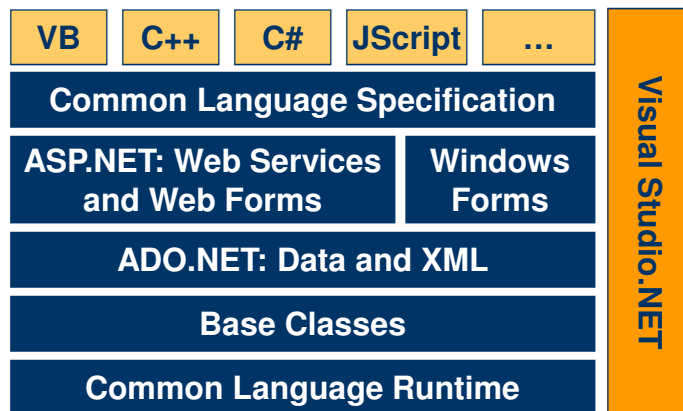
## Architecture

- ◆ ASP.NET is built upon
  - .NET Framework
  - Internet Information Server (IIS)

# ASP.NET Overview Architecture

- ◆ Internet Information Server (IIS)
  - IIS MMC Snap-In (Internet Services Manager)
    - Tool to manage IIS
  - Virtual Directories
    - Provides a mapping between URL and file path
    - E.g., on my machine the URL:  
`http://localhost/cs5950`  
maps to the file path:  
`c:\cs5950`

# ASP.NET Overview Architecture





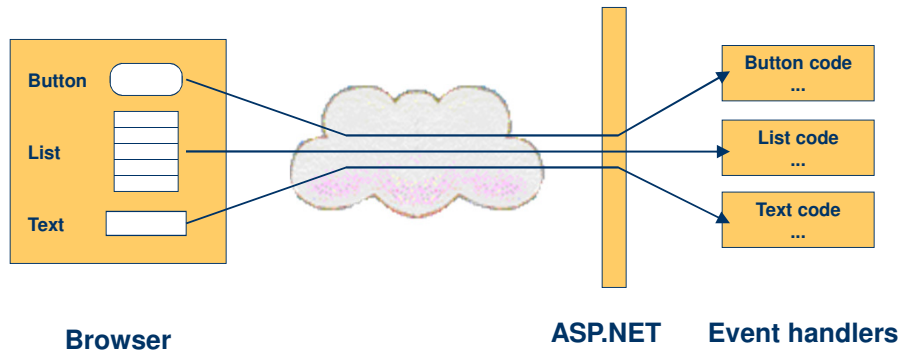
## Outline

- ◆ Background
- ◆ ASP.NET Overview
- ◆ **Programming Model**
- ◆ Programming Basics
- ◆ Server Controls
- ◆ Data Binding
- ◆ Conclusion

## Programming Model Controls and Events

- ◆ Server-side programming model
- ◆ Based on controls and events
  - Just like Visual Basic
  - Not “data in, HTML out”
- ◆ Higher level of abstraction than ASP
- ◆ Requires less code
- ◆ More modular, readable, and maintainable

## Programming Model Controls and Events



## Programming Model ASP.NET Object Model

- ◆ User code executes on the web server in page or control event handlers
- ◆ Controls are objects, available in server-side code
  - Derived from `System.Web.UI.Control`
- ◆ The web page is an object too
  - Derived from `System.Web.UI.Page` which is a descendant of `System.Web.UI.Control`
  - A page can have methods, properties, etc.

## Programming Model Postbacks

- ◆ A postback occurs when a page generates an HTML form whose values are posted back to the same page
- ◆ A common technique for handling form data
- ◆ In ASP and other server-side technologies the state of the page is lost upon postback...
- ◆ Unless you explicitly write code to maintain state
- ◆ This is tedious, bulky and error-prone

## Programming Model Postbacks Maintain State

- ◆ By default, ASP.NET maintains the state of all server-side controls during a postback
- ◆ Can use `method="post"` or `method="get"`
- ◆ Server-side control objects are automatically populated during postback
- ◆ No state stored on server
- ◆ Works with all browsers

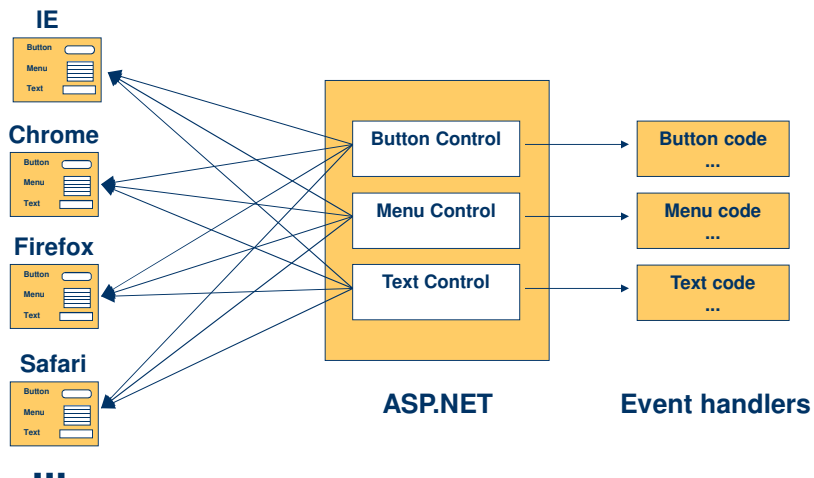
## Programming Model Server-side Controls

- ◆ Multiple sources of controls
  - Built-in
  - 3<sup>rd</sup> party
  - User-defined
- ◆ Controls range in complexity and power: button, text, drop down, calendar, data grid, ad rotator, validation
- ◆ Can be populated via data binding

## Programming Model Automatic Browser Compatibility

- ◆ Controls can provide automatic browser compatibility
- ◆ Can target UpLevel or DownLevel browsers
  - UpLevel browsers support additional functionality, such as JavaScript and DHTML
  - DownLevel browsers support HTML

## Programming Model Automatic Browser Compatibility



## Programming Model Code-behind pages

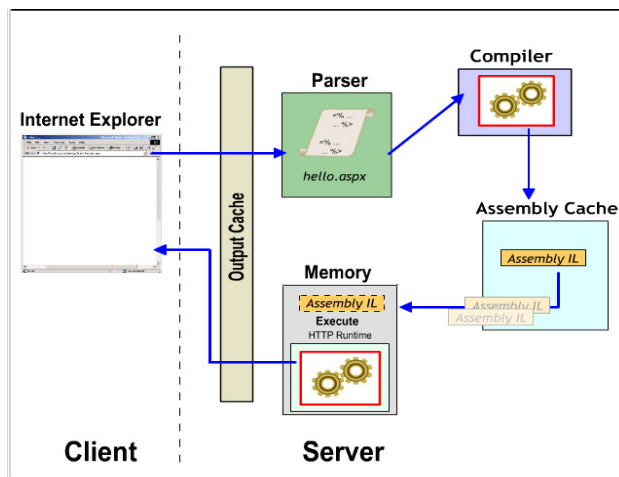
- ◆ Two styles of creating ASP.NET pages
  - Controls and code in .aspx file
  - Controls in .aspx file, code in code-behind page
    - Supported in Visual Studio.NET
- ◆ Code-behind pages allow you to separate the user interface design from the code
  - Allows programmers and designers to work independently

```
<%@ Codebehind="WebForm1.cs"  
Inherits=WebApplication1.WebForm1" %>
```

## Programming Model Automatic Compilation

- ◆ Just edit the code and hit the page
- ◆ ASP.NET will automatically compile the code into an assembly
- ◆ Compiled code is cached in the CLR Assembly Cache
- ◆ Subsequent page hits use compiled assembly
- ◆ If the text of the page changes then the code is recompiled
  - Works just like ASP: edit, save and run

## Programming Model Automatic Compilation



## Outline

- ◆ Background
- ◆ ASP.NET Overview
- ◆ Programming Model
- ◆ **Programming Basics**
- ◆ Server Controls
- ◆ Data Binding
- ◆ Conclusion

## Programming Basics Page Syntax

- ◆ The most basic page is just static text
  - Any HTML page can be renamed .aspx
- ◆ Pages may contain:
  - Directives: `<%@ Page Language="C#" %>`
  - Server controls: `<asp:Button runat="server">`
  - Code blocks: `<script runat="server">...</script>`
  - Data bind expressions: `<%# %>`
  - Server side comments: `<%-- --%>`
  - Render code: `<%= %>` and `<% %>`
    - Use is discouraged; use `<script runat=server>` with code in event handlers instead

## Programming Basics

### The Page Directive

- ◆ Lets you specify page-specific attributes, e.g.
  - `AspCompat`: Compatibility with ASP
  - `Buffer`: Controls page output buffering
  - `CodePage`: Code page for this .aspx page
  - `ContentType`: MIME type of the response
  - `ErrorMessage`: URL if unhandled error occurs
  - `Inherits`: Base class of Page object
  - `Language`: Programming language
  - `Trace`: Enables tracing for this page
  - `Transaction`: COM+ transaction setting
- ◆ Only one page directive per .aspx file

## Programming Basics

### Server Control Syntax

- ◆ Controls are declared as HTML tags with `runat="server"` attribute

```
<input type=text id=text2 runat="server" />
<asp:calendar id=myCal runat="server" />
```
- ◆ **Tag** identifies which type of control to create
  - Control is implemented as an ASP.NET class
- ◆ The **id** attribute provides programmatic identifier
  - It names the instance available during postback
  - Just like Dynamic HTML



## Programming Basics Server Control Properties

- ◆ Tag attributes map to control properties

```
<asp:button id="c1" Text="Foo" runat="server">  
<asp:ListBox id="c2" Rows="5" runat="server">
```

- ◆ Tags and attributes are case-insensitive
- ◆ Control properties can be set programmatically

```
c1.Text = "Foo";  
c2.Rows = 5;
```

## Programming Basics Maintaining State

- ◆ By default, controls maintain their state across multiple postback requests
  - Implemented using a hidden HTML field: `__VIEWSTATE`
  - Works for controls with input data (e.g. `TextBox`, `CheckBox`), non-input controls (e.g. `Label`, `DataGrid`), and hybrids (e.g. `DropDownList`, `ListBox`)
- ◆ Can be disabled per control or entire page
  - Set `EnableViewState="false"`
  - Lets you minimize size of `__VIEWSTATE`

## Programming Basics

### Maintaining State

- ◆ Example: MaintainingState.aspx



## Programming Basics

### Server Code Blocks

- ◆ Server code lives in a script block marked `runat="server"`

```
<script language="C#" runat=server>  
<script language="VB" runat=server>  
<script language="JScript" runat=server>
```

- ◆ Script blocks can contain
  - Variables, methods, event handlers, properties
  - They become members of a custom Page object

## Programming Basics Page Events

- ◆ Pages are structured using events
  - Enables clean code organization
  - Avoids the “Monster IF” statement
  - Less complex than ASP pages
- ◆ Code can respond to page events
  - e.g. Page\_Load, Page\_Unload
- ◆ Code can respond to control events
  - Button1\_Click
  - Textbox1\_Changed

## Programming Basics Page Event Lifecycle



## Programming Basics

### Page Loading

- ◆ Page\_Load fires at beginning of request after controls are initialized
  - Input control values already populated

```
protected void Page_Load(Object s, EventArgs e) {  
    message.Text = textbox1.Text;  
}
```

## Programming Basics

### Page Loading

- ◆ Page\_Load fires on every request
  - Use Page.IsPostBack to execute conditional logic
  - If a Page/Control is maintaining state then need only initialize it when IsPostBack is false

```
protected void Page_Load(Object s, EventArgs e) {  
    if (! Page.IsPostBack) {  
        // Executes only on initial page load  
        Message.Text = "initial value";  
    }  
    // Rest of procedure executes on every request  
}
```

## Programming Basics Server Control Events

- ◆ Change Events
  - By default, these execute only on next action event
  - E.g. `onTextChanged`, `onCheckedChanged`
  - Change events fire in random order
- ◆ Action Events
  - Cause an immediate postback to server
  - E.g. `onClick`
- ◆ Works with any browser
  - No client script required, no applets, no ActiveX® Controls!

## Programming Basics Wiring Up Control Events

- ◆ Control event handlers are identified on the tag

```
<asp:button onclick="btn1_click" runat=server>  
<asp:textbox onchange="text1_changed" runat=server>
```

- ◆ Event handler code

```
protected void btn1_Click(Object s, EventArgs e) {  
    Message.Text = "Button1 clicked";  
}
```

## Programming Basics

### Event Arguments

- ◆ Events pass two arguments:
  - The sender, declared as type `object`
    - Usually the object representing the control that generated the event
    - Allows you to use the same event handler for multiple controls
  - Arguments, declared as type `EventArgs`
    - Provides additional data specific to the event
    - `EventArgs` itself contains no data; a class derived from `EventArgs` will be passed

## Programming Basics

### Page Unloading

- ◆ `Page_Unload` fires after the page is rendered
  - Don't try to add to output
- ◆ Useful for logging and clean up

```
protected void Page_Unload(Object s, EventArgs e) {  
    MyApp.LogPageComplete();  
}
```

## Programming Basics

### Import Directive

- ◆ Adds code namespace reference to page
  - Avoids having to fully qualify .NET types and class names
  - Equivalent to the C# using directive

```
<%@ Import Namespace="System.Data" %>  
<%@ Import Namespace="System.Net" %>  
<%@ Import Namespace="System.IO" %>
```

## Programming Basics

### Page Class

- ◆ The Page object is always available when handling server-side events
- ◆ Provides a large set of useful properties and methods, including:
  - **Application**, **Cache**, **Controls**, **EnableViewState**, **EnableViewStateMac**, **ErrorPage**, **IsPostBack**, **IsValid**, **Request**, **Response**, **Server**, **Session**, **Trace**, **User**, **Validators**
  - **DataBind()**, **LoadControl()**, **MapPath()**, **Validate()**

## Outline

- ◆ Background
- ◆ ASP.NET Overview
- ◆ Programming Model
- ◆ Programming Basics
- ◆ **Server Controls**
- ◆ Data Binding
- ◆ Conclusion

## Server Controls

- ◆ ASP.NET ships with ~50 built-in controls
- ◆ Organized into logical families
  - HTML controls
    - Controls / properties map 1:1 with HTML
  - Web controls
    - Richer functionality
    - More consistent object model



## Server Controls HTML Controls

- ◆ Work well with existing HTML designers
- ◆ Properties map 1:1 with HTML  
`table.bgcolor = "red";`
- ◆ Can specify client-side event handlers
- ◆ Good when quickly converting existing pages
- ◆ Derived from  
`System.Web.UI.HtmlControls.HtmlControl`
- ◆ Supported controls have custom class,  
others derive from `HtmlGenericControl`

## Server Controls HTML Controls

- ◆ Supported controls
  - `<a>`
  - `<img>`
  - `<form>`
  - `<table>`
  - `<tr>`
  - `<td>`
  - `<th>`
  - `<select>`
  - `<textarea>`
  - `<button>`
  - `<input type=text>`
  - `<input type=file>`
  - `<input type=submit>`
  - `<input type=button>`
  - `<input type=reset>`
  - `<input type=hidden>`

## Server Controls HTML Controls

- ◆ Example: HTMLControls.aspx
  - Basic page lifecycle with HTML Controls



## Server Controls Web Controls

- ◆ Consistent object model

```
Label1.BackColor = color.Red;
Table.BackColor = color.Blue;
```
- ◆ Richer functionality
  - E.g. AutoPostBack, additional methods
- ◆ Automatic uplevel/downlevel support
  - E.g. validation controls
- ◆ Strongly-typed; no generic control
  - Enables better compiler type checking

## Server Controls Web Controls

- ◆ Web controls appear in HTML markup as namespaced tags
- ◆ Web controls have an `asp:` prefix

```
<asp:button onclick="button1_click" runat=server>  
<asp:textbox onchange="text1_changed" runat=server>
```

- ◆ Defined in the `System.Web.UI.WebControls` namespace
- ◆ This namespace is automatically mapped to the `asp:` prefix

## Server Controls Web Controls

- ◆ Web Controls provide extensive properties to control display and format, e.g.
  - Font
  - BackColor, ForeColor
  - BorderColor, BorderStyle, Borderwidth
  - Style, CssClass
  - Height, width
  - visible, Enabled

## Server Controls Web Controls

- ◆ Four types of Web Controls
  - Intrinsic controls
  - List controls
  - Rich controls
  - Validation controls

## Server Controls Intrinsic Controls

- ◆ Correspond to HTML controls
- ◆ Supported controls
  - `<asp:button>`
  - `<asp:imagebutton>`
  - `<asp:linkbutton>`
  - `<asp:hyperlink>`
  - `<asp:textbox>`
  - `<asp:checkbox>`
  - `<asp:radiobutton>`
  - `<asp:image>`
  - `<asp:label>`
  - `<asp:panel>`
  - `<asp:table>`
- ◆ Specify `AutoPostBack=true` to make change events cause a postback

## Server Controls

### List Controls

- ◆ Controls that handle repetition
- ◆ Supported controls
  - `<asp:dropdownlist>`
  - `<asp:listbox>`
  - `<asp:radiobuttonlist>`
  - `<asp:checkboxlist>`
  - `<asp:repeater>`
  - `<asp:datalist>`
  - `<asp:datagrid>`

## Server Controls

### CheckBoxList & RadioButtonList

- ◆ Provides a collection of check box or radio button controls
- ◆ Can be populated via data binding

```
<asp:CheckBoxList id=Check1 runat="server">  
  <asp:ListItem>Item 1</asp:ListItem>  
  <asp:ListItem>Item 2</asp:ListItem>  
  <asp:ListItem>Item 3</asp:ListItem>  
  <asp:ListItem>Item 4</asp:ListItem>  
  <asp:ListItem>Item 5</asp:ListItem>  
</asp:CheckBoxList>
```

## Server Controls

### Intrinsic & Simple List Controls

- ◆ Example: WebControls.aspx
  - Assorted intrinsic and list controls with AutoPostBack



## Server Controls

### Rich Controls

- ◆ Custom controls with rich functionality
- ◆ Supported Controls
  - `<asp:calendar>`
  - `<asp:adrotator>`
- ◆ More will be added
- ◆ 3<sup>rd</sup> party controls are coming
- ◆ Example: RichControls.aspx



## Server Controls Validation Controls

- ◆ Rich, declarative validation
- ◆ Validation declared separately from input control
- ◆ Extensible validation framework
- ◆ Supports validation on client and server
  - Automatically detects uplevel clients
  - Avoids roundtrips for uplevel clients
- ◆ Server-side validation is always done
  - Prevents users from spoofing Web Forms

## Server Controls Validation Controls

- ◆ `<asp:RequiredFieldValidator>`
  - Ensures that a value is entered
- ◆ `<asp:RangeValidator>`
  - Checks if value is within minimum and maximum values
- ◆ `<asp:CompareValidator>`
  - Compares value against constant, another control or data type
- ◆ `<asp:RegularExpressionValidator>`
  - Tests if value matches a predefined pattern
- ◆ `<asp:CustomValidator>`
  - Lets you create custom client- or server-side validation function
- ◆ `<asp:ValidationSummary>`
  - Displays list of validation errors in one place

## Server Controls Validation Controls

- ◆ Validation controls are derived from `System.Web.UI.WebControls.BaseValidator`, which is derived from the `Label` control
- ◆ Validation controls contain text which is displayed only if validation fails
- ◆ **Text property is displayed at control location**
- ◆ **ErrorMessage is displayed in validation summary**

## Server Controls Validation Controls

- ◆ Validation controls are associated with their target control using the `ControlToValidate` property

```
<asp:TextBox id=TextBox1 runat=server />  
  
<asp:RequiredFieldValidator id="Req1"  
  ControlToValidate="TextBox1"  
  Text="Required Field" runat=server />
```

- ◆ Can create multiple validation controls with the same target control



## Server Controls Validation Controls

- ◆ `Page.IsValid` indicates if all validation controls on the page succeed

```
void Submit_Click(object s, EventArgs e) {  
    if (Page.IsValid) {  
        Message.Text = "Page is valid!";  
    }  
}
```

## Server Controls Validation Controls

- ◆ `Display` property controls layout
  - `Static`: fixed layout, display won't change if invalid
  - `Dynamic`: dynamic layout
  - `None`: no display; can still use `ValidationSummary` and `Page.IsValid`
- ◆ `Type` property specifies expected data type: `Currency`, `Date`, `Double`, `Integer`, `String`

## Server Controls Validation Controls

- ◆ Can force down-level option
  - Only server-side validation

```
<% @ Page Language="c#"
      ClientTarget="DownLevel1" %>
```

## Server Controls Validation Controls

- ◆ Example: ValidationControls.aspx

