Unit 6

7.0 Web Controls

(Text Book Chapter 06)

7.0 Web Controls

We have, buy this time investigated what "HTML Server Controls" are. Having seen this model of server controls, you might wonder why you need additional set of controls called web controls. HTML controls are much more limited than server controls need to be. For example, every HTML control corresponds directly to a single HTML element. Web controls, on the other hand, have no such restriction; they can switch from one element to another depending on how you're using them, or they can render themselves using a complex combination of multiple elements.

They provide a rich user interface: A web control is programmed as an object but doesn't necessarily correspond to a single element in the final HTML page. For example, you might create a single **Calendar** or **GridView** control, which will be rendered as dozens of HTML elements in the final page. When using ASP.NET programs, you don't need to know anything about HTML. The control creates the required HTML tags for you.

They provide a consistent object model: HTML is full of quirks and idiosyncrasies. For example, a simple text box can appear as one of three elements, including `<textarea>`, `<input type="text">`, and `<input type="password">`. With web controls, these three elements are consolidated as a single **TextBox** control. **Depending on the properties you set,** the underlying HTML element that ASP.NET renders may differ. Similarly, the names of properties don't follow the HTML attribute names. For example, controls that display text, whether it's a caption or a text box that can be edited by the user, expose a Text property.

They tailor their output automatically: ASP.NET server controls can detect the type of browser and automatically adjust the HTML code they write to take advantage of features such as support for JavaScript. You don't need to know about the client because ASP.NET handles that layer and automatically uses the best possible set of features. This feature is known as adaptive rendering.

➢7.0 Web Controls

They provide high-level features: You'll see that web controls allow you to access additional events, properties, and methods that don't correspond directly to typical HTML controls. ASP.NET implements these features by using a combination of tricks.

To master ASP.NET development, **you need to become comfortable with these user-interface ingredients and understand their abilities.** HTML Server controls, on the other hand, are less important for web forms development. You'll only use them if you're migrating an existing HTML page to the ASP.NET world, or if you need to have fine-grained control over the HTML code that will be generated and sent to the client.

7.1 Basic Web Control Classes

If you've ever created a Windows application before, you're probably familiar with the basic set of standard controls, including labels, buttons, and text boxes. ASP.NET provides web controls for all these standbys. (And if you've created .NET Windows applications, you'll notice that the class names and properties have many striking similarities, which are designed to make it easy to transfer the experience you acquire in one type of application to another.)

Figure 7.1-1 lists the basic control classes and the HTML elements they generate. Some controls (such as Button and TextBox) can be rendered as different HTML elements. In this case, ASP.NET uses the element that matches the properties you've set. Also, some controls have no real HTML equivalent. For example, when the CheckBoxList and RadioButtonList controls render themselves, they may output a <table> that contains multiple HTML check boxes or radio buttons. ASP.NET exposes them as a single object on the server side for convenient programming, thus illustrating one of the primary strengths of web controls.

7.0 Web Controls

Control Class	Underlying HTML Element
Label	
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">
Image	<img> ListBox <select size="X"> where X is the number of rows that are visible at once
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags
BulletedList	An ordered list (numbered) or unordered list (bulleted)
Panel	
Table, TableRow, and TableCell	<table>, <tr>, and <td> or <th>

Figure 7.1-1 Web Controls & Their Underlying Classes

The table above omits some of the more specialized controls used for data, navigation, security, and web portals. You'll see these controls in future chapters.

➢7.0 Web Controls

7.1 Web Control Tags

ASP.NET tags have a special format. They always begin with the prefix **asp:** followed by the **class name**. If there is no closing tag, the tag must end with />. (This syntax convention is borrowed from XML. Each attribute in the tag corresponds to a control property, except for the **runat="server"** attribute, which signals that the control should be processed on the server. asp: controls must be defined in files having extension **.aspx**.

The general tag for creating a Web control is:

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

eg: A Text Box Web Control Tag:

<asp:TextBox ID="txt" runat="server" />

When a client requests the **.aspx** page that contains the **asp: TextBox** control, the following HTML is returned.

```
<input type="text" ID="txt" name="txt" />
```

The name is a special attribute that ASP.NET uses to track the control.

Alternatively, you could place some text in the TextBox, set its size, make it read-only, and change the background color. All these actions have define d properties. For example, the TextBox.TextMode property allows you to specify **SingleLine** (the default), **MultiLine** (for a <textarea> type of control), or **Password** (for an input control that displays bullets to hide the true value). You can adjust the color using the **BackColor** and **ForeColor** properties. And you can tweak the size of the TextBox in one of two ways—either using the Rows and Columns properties (for the pure HTML approach) or using the Height and Width properties (for the style-based approach). Both have the same result.

Here's one example of a customized TextBox:

```
<asp:TextBox ID="txt" BackColor="Yellow" Text="Hello World"
 ReadOnly="True" TextMode="MultiLine" Rows="5"
    runat="server" />
```

The resulting HTML (See below) uses the <textarea> element and sets all the required attributes (such as rows and readonly) and the style attribute (with the Background color). It also sets the cols attribute with the default width of 20 columns, even though you didn't explicitly set the TextBox.Columns property:

```
<textarea name="txt" rows="5" cols="20" readonly="readonly"
    id="txt" style="background-color:Yellow;">Hello
    World</textarea>
```

Figure shows the result when you execute this tag.

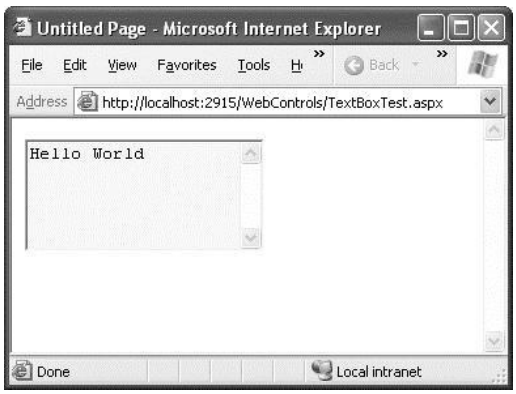

Figure 7.1-2

Clearly, it's easy to create a web control tag. It doesn't require any understanding of HTML. However, you will need to understand the control class and the properties that are available to you. The .aspx layout portion of a web page tolerates different capitalization for tag names, property names, and enumeration values. For example, the following two tags are equivalent, and both will be interpreted correctly by the ASP.NET engine, even though their case differs:

```
<asp:Button ID="Button1" runat="server"
 Enabled="False" Text="Button" Font-Size="XX-Small" />
<asp:button ID="Button2" runat="server" Enabled="false"
    tExT="Button" Font-Size="xx-small" />
```

This design was adopted to make .aspx pages behave more like ordinary HTML web pages, which ignore case completely. However, you can't use the same looseness in the tags that apply settings in the web.config file or the machine.config file. Here, case must match exactly the machine.config.

7.2 Common features

Mostly **all Web Server controls inherit from a common base class, namely the WebControl** class defined in the S**ystem.Web.UI.WebControls namespace.** This base class implements a set of common properties that all Web Server controls have: **Attributes, CSSClass, BackColor, Enabled, Visible, Width, etc.** Web server controls that do not inherit from the WebControl class include **Literal**, **Place Holde**r, **Repeater**, and **Xml**.

All Web Controls on a Web Form must have a name - specified using the "ID“ attribute. You use the ID attribute, the name of the component, to access its properties and methods.

7.2.1 Web Control Classes

Following figure shows a list of commonly used Web Controls.

Web Server Control	Description
asp:Button	Displays a push button
asp:Calendar	Displays a calendar
asp:CalendarDay	A day in a calendar control
asp:CheckBox	Displays a check box
asp:CheckBoxList	Creates a multi-selection check box group
asp:DropDownList	Creates a drop-down list
asp:HyperLink	Creates a hyperlink
asp:Image	Displays an image
asp:ImageButton	Displays a clickable image
asp:Label	Displays static content which is programmable (lets you apply styles to its content)
asp:LinkButton	Creates a hyperlink button
asp:ListBox	Creates a single- or multi-selection drop-down list
asp:Panel	Provides a container for other controls
asp:RadioButton	Creates a radio button
asp:RadioButtonList	Creates a group of radio buttons
asp:Table	Creates a table
asp:TableCell	Creates a table cell
asp:TableRow	Creates a table row
asp:TextBox	Creates a text box

Figure 7.2-1 Common Web Server Controls

7.3 Web Server Control Categories

7.3.1 Intrinsic controls

Intrinsic controls render to simple HTML elements. For example, a Button Web Server controls renders to an <input> element, but provides additional server-side functionality for a developer

7.3.2 Validation controls

Validation controls provide a way to reduce the number of server round-trips by adding client side validation code

7.3.3 Rich controls

like **Calendar** or **DataGrid** provide a rich user interface for particular tasks. They both render to quite complex table elements. What's more **DataGrid** even enables editing data from a database ... but we'll leave this discussion for some other time.

7.4 Web Control Properties

All web controls have these standard properties. Here is a list of standard properties and their descriptions (Figure 7.3-1)

Property	Description
BorderStyle	The BorderStyke property is used to set or return the border style of a control.
Controls	The collection of a control's child controls.
Enabled	Enables the control
EnableViewState	Indicates whether a control maintains its view state across round trips. If a parent control does not maintain its view state, the view state for its child controls is automatically not maintained.
Font-Size	The size of the font for the control. Variations –Bold, -Italic –Names etc…
Height	The height of the control.
Width	The width of the control.
Page	The page that contains the control.
Parent	The control whose Controls collection a control belongs to. (Control A is a parent of control B if B is an element of A.Controls).
TableIndex	Returns the DataTable at the specified index.
ToolTip	Sets or return the text that appears when the user rests the mouse pointer over a control.
Visible	Determines whether a control is visible on a page.

Figure 7.4-1 Web Controls common Properties

Simple example of using properties.

```
!Example 7.4-1a
<form id="Form2" runat="server">
   <asp:Table ID="Table1" runat="server" BorderStyle="dotted" BorderWidth="5"
GridLines="vertical">
      <asp:TableRow>
         <asp:TableCell>Hello</asp:TableCell>
         <asp:TableCell>World</asp:TableCell>
      </asp:TableRow>
   </asp:Table>
</form>
```

Example 7.4-1a Using Properties

Figure 7.41

```
<form id="Form2" runat="server">
   <asp:Table ID="Table1" runat="server" BorderStyle="dotted" BorderWidth="5"
GridLines="vertical">
      <asp:TableRow>
         <asp:TableCell Font-Size="14"
                        ForeColor="Red">Hello</asp:TableCell>
         <asp:TableCell Font-Size="8"
                        ForeColor="Green">World</asp:TableCell>
      </asp:TableRow>
   </asp:Table>
</form>
```

Example 7.4-1b Using Properties

Figure 7.4-2

7.4.1 asp:Table Control Properties

Web Control asp:Table is important and worth discussing. The Table control is used in conjunction with the TableCell control and the TableRow control. Table control has following properties (Figure 7.4-3).

Table Property	Description
BackImageUrl	A URL to an image to use as an background for the table
Caption	The caption of the table
CaptionAlign	The alignment of the caption text
CellPadding	The space between the cell walls and contents
CellSpacing	The space between cells
GridLines	The gridline format in the table
HorizontalAlign	The horizontal alignment of the table in the page
Rows	A collection of rows in the Table
runat	Specifies that the control is a server control. Must be set to "server"

Figure 7.4-3 asp:Table Properties

➢7.0 Web Controls

Couple of simple examples are shown below.

```
!Example 7.4-2
<form runat=server>
<asp:Table runat="server" CellPadding="5"
GridLines="horizontal" HorizontalAlign="Center">
  <asp:TableRow>
    <asp:TableCell>1</asp:TableCell>
    <asp:TableCell>2</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>3</asp:TableCell>
    <asp:TableCell>4</asp:TableCell>
  </asp:TableRow>
</asp:Table>
<br />
<asp:Table runat="server" CellPadding="5"
GridLines="vertical" HorizontalAlign="Center">
  <asp:TableRow>
    <asp:TableCell>1</asp:TableCell>
    <asp:TableCell>2</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>3</asp:TableCell>
    <asp:TableCell>4</asp:TableCell>
  </asp:TableRow>
</asp:Table>
</form>
```

Example 7.4-2 Using asp:Table Properties

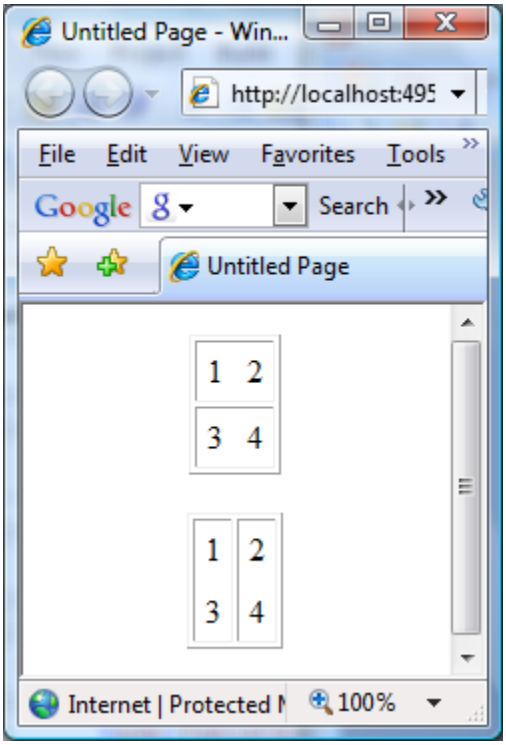

Figure 7.4-3

7.5 Web Control Events and Autopostback

The **AutoPostBack** property is used to set or return whether or not an automatic post back occurs when the user presses "ENTER" or "TAB" in the TextBox control.

What is means is this. Normally, in a form, you enter all inputs on text boxes etc, an then you click a button to "Submit". Then a post back happens. Which is the result after you clicked the button. But in some cases you want To **AutoPostBack** while you type or soon after you finish typing a specific TextBox. For example, in a TextBox field if you want the server to process what you entered ONLY ON THAT TEXTBOX soon after you finished typing, without waiting a "Submit" button to be pressed, then the AutoPost must be turned on.

These situation happens if you want to make sure that a TextBox allowed to type only 'Alpha' characters. If the user entered a 'digit' character immediately a message pops up saying that you typed a 'Digit' instead of an 'Alpha'. Some people use this feature in user ID/password fields.

AutoPostBack property is set to TRUE the automatic post back is enabled, otherwise FALSE. Default is FALSE. The syntax is shown below…

```
<asp:TextBox AutoPostBack="TRUE|FALSE" runat="server"/>
```

Example.
The following example sets the **AutoPostBack** mode to "TRUE":

```
<form runat="server">
  <asp:TextBox id="tb1" runat="server" AutoPostBack="TRUE" />
</form>
```

➢7.0 Web Controls

In the following example, page first displays like in Figure 7.5-1, Then you type " m" and then press The return key [Not the Submit Button].
You see the result on Figure 7.5-2.

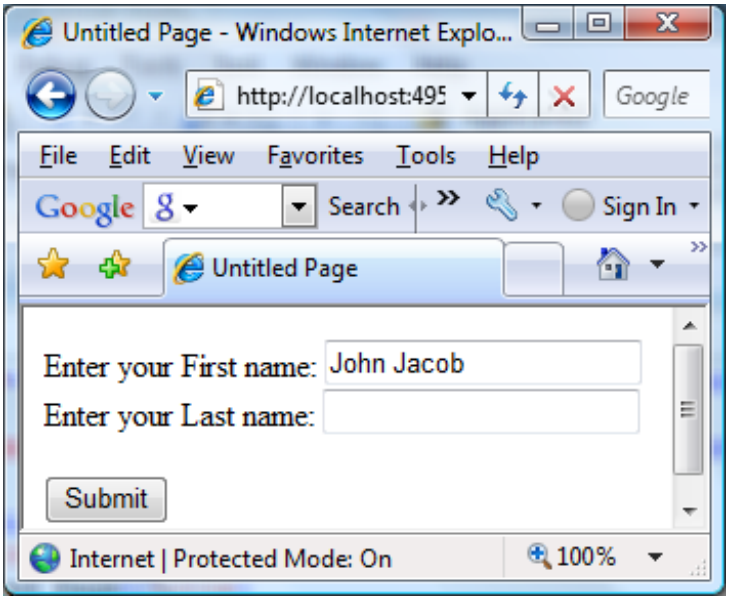

Figure 7.5-1

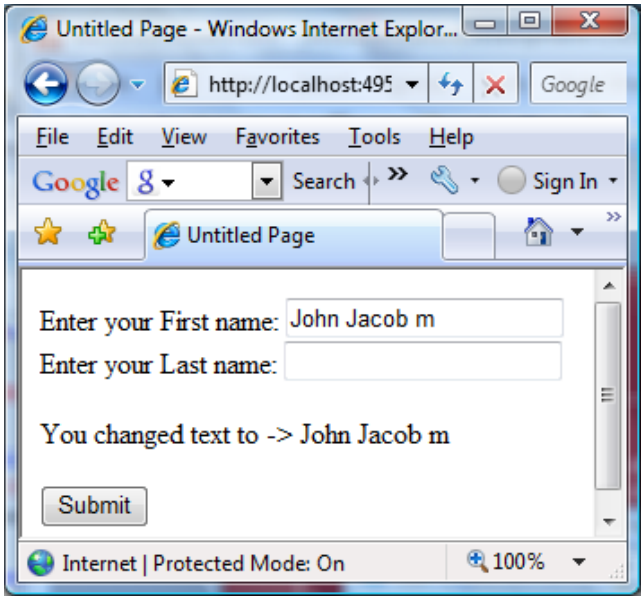

Figure 7.5-2

7.4.1 Inside of PostBack – How does it work?
We will take a closer look at how ASP.NET pages post back to themselves, and how to customize this feature in our web applications.

```
function __doPostBack(eventTarget, eventArgument)
```

When form fields and other controls can be declared to run on the server, and the server simply posts the page back to itself and performs all the validation, display and actions. Our life as web developers has become a million times better.

But how exactly is this done? For this an internal C#script method called the `_doPostBackEvent()` is used. See pages 195-199 for further details.

7.6 Page Life Cycle

To understand how web control events work, you need to have a solid understanding of the page lifecycle. Consider what happens when a user changes a control that has the **AutoPostBack** property set to **true:**

1. On the client side, the JavaScript **__doPostBack** function is invoked, and the page is resubmitted to the server.
2. ASP.NET re-creates the Page object using the .aspx file.
3. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
4. The **Page.Load** event is fired.
5. The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
6. The **Page.PreRender** event fires, and the page is rendered (transformed from a set of objects to an HTML page).
7. Finally, the **Page.Unload** event is fired.
8. The new page is sent to the client.

To watch these events in action, it helps to create a simple event tracker application. All this application does is write a new entry to a list control every time one of the events it's monitoring occurs. This allows you to see the order in which events are triggered.

Figure 7.6-1 shows what the window looks like after it's been loaded once, posted back (when the text box content was changed), and posted back again (when the check box state was changed)

➢7.0 Web Controls

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs"
Inherits="EventTracker._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></title>
</head>
<body>
  <form id="form1" runat="server">
   <div>
    <h1>Controls being monitored for change events:</h1>
     <asp:TextBox ID="txt" runat="server“
     AutoPostBack="true" OnTextChanged="CtrlChanged" />
     <br /><br />
     <asp:CheckBox ID="chk" Text="Check/Uncheck“
       runat="server" AutoPostBack="true“
        OnCheckedChanged="CtrlChanged"/>
      <br /><br />
      <asp:RadioButton ID="opt1" Text="Option 1"
                  runat="server" GroupName="Sample"
                  AutoPostBack="true"
           OnCheckedChanged="CtrlChanged"/>
      <asp:RadioButton ID="opt2" Text="Option 2"
           runat="server" GroupName="Sample"
           AutoPostBack="true"
           OnCheckedChanged="CtrlChanged"/>
       <h1>List of events:</h1>
     <asp:ListBox ID="lstEvents" runat="server"
            Width="355px" Height="150px" /><br />
     <br /><br /><br />
   </div>
  </form>
</body>
</html>
```

Example 7.6-1 a

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace EventTracker
{
   public partial class _Default : System.Web.UI.Page
    {
       protected void Page_Load(object sender,
                                          EventArgs e)
       { Log("<< Page_Load >>");  }

       protected void Page_PreRender(object sender,
                                          EventArgs e)
       {
          // When the Page.PreRender event occurs,
          //it is too late  to change the list.
          Log("Page_PreRender");
       }

       protected void CtrlChanged(Object sender,
                                          EventArgs e)
       { // Find the control ID of the sender.
          // This requires converting the Object
          //type into a Control class.
          string ctrlName = ((Control)sender).ID;
          Log(ctrlName + " Changed");
       }

       private void Log(string entry)
       {
          lstEvents.Items.Add(entry);
          // Select the last item to scroll the list so the
          //most recent entries are visible.
          lstEvents.SelectedIndex =
                              lstEvents.Items.Count - 1;

       }
    }
}
```

Example 7.6-1 b

➢7.0 Web Controls

The following points are worth noting about this code:The code writes to the ListBox using a private Log() method. The Log() method adds the text and automatically scrolls to the bottom of the list each time a new entry is added, thereby ensuring that the most recent entries remain visible.

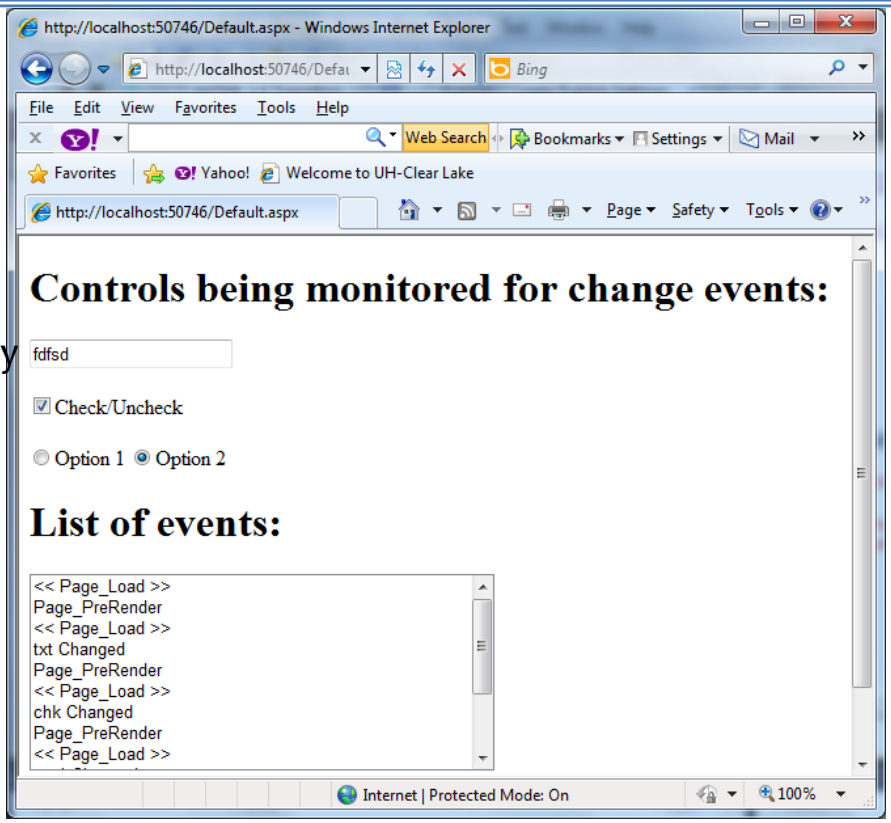

Figure 7.6-1

All the change events are handled by the same method, CtrlChanged().If you look carefully at the .aspx you'll notice see that each input control connects its monitored event to the CtrlChanged() method. The event handling code in the CtrlChanged() method uses the source parameter to find out what control sent the event, and it incorporates that information in the log string.

The page includes event handlers for the Page.Load and Page.PreRender events. As with all page events, these event handlers are connected by method name. That means to add the event handler for the **Page.PreRender** event, you simply need to add a method named Page_PreRender(), (See example code above).

PS: Finally, worth looking at the Greeting Card example in text book pages 201-210 (Instructor do a demo)