Basic Web Controls

Contents

[1 Introduction to Web Controls 1](#_Toc127101422)

[2 Overview of Basic Web Controls 3](#_Toc127101423)

[3 Introduction to List Controls 5](#_Toc127101424)

[3.1 Example 6](#_Toc127101425)

[3.2 SelectedIndexChanged Event 6](#_Toc127101426)

[3.3 Most common scenario: No Event Handler 7](#_Toc127101427)

[3.4 Other Scenarios 7](#_Toc127101428)

[3.5 A fuller description of the List Controls: 9](#_Toc127101429)

[3.6 Value Must be Unique 10](#_Toc127101430)

[3.7 Adding & Removing ListItems 11](#_Toc127101431)

[3.8 Removing ListItems – Approaches that don’t Work 11](#_Toc127101432)

[Appendix 1 n/a 12](#_Toc127101433)

# Introduction to Web Controls

1. An *ASP.NET server control* (also called: *web control* or *control*) is a tag written inside a web page that represents a server-side object that is programmable and is used for displaying information on a web page, collecting information from a user, or allowing a user to cause some action by interacting with it.
2. For example, an *ASP.NET Button* could be defined like this in a web page on the server.

<asp:Button ID="btnPurchase" runat="server" onclick="btnPurchase\_Click" Text="Purchase" />

The *runat* attribute specifies that the control is run on the server. When the server delivers the page to a browser, it is converted into a HTML element:

<input type="submit" name="btnPurchase" value="Purchase" id="btnPurchase" />

All web controls are translated to HTML before being sent to the server.

1. Behind the scenes there is a lot going on. The server creates JavaScript code and sends it to the browser so that when this button is pressed by a user:

* The page is posted back to the server
* When the page is posted back to the server, a *Button* instance is created (on the server) and can be programmed. For instance, we can change the *Text* on the button
* The *btnPurchase\_Click* event handler is called on the server

1. Essentially what Microsoft (MS) has done is create a virtual programming environment where we can create a user interface and write event handlers without having to think about how to do this in HTML and JavaScript. In other words, it allows us to us a more robust programming environment. When a page is requested from the server, the ASP.NET engine processes the page and renders it in HTML and provides the JavaScript to support it.
2. We will study these web controls: *Label, Button, TextBox, DropDownList, ListBox, CheckBoxList, RadioButtonList. Calendar,* and *Panel*

# Overview of Basic Web Controls

Web controls have properties, methods, and events. Below, we list the most common properties, methods, and events for each of the controls mentioned above.

1. [WebControl](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.webcontrol.aspx) – This is the super class for all controls

Properties:

1. Enabled – When *true,* this control can be interacted with. When *false* it is visible, but grayed out.
2. Visible – When *true,* the control is displayed on the web page. When *false,* it is not visible.

We have omitted style properties (*BackColor, BorderColor, BorderSyle, BorderWidth, Font, ForeColor, Height, SkinID, ToolTip, Width,* and others) which all controls have.

Methods:

1. Focus – Gives the control focus.
2. Control: [Button](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx)

|  |  |
| --- | --- |
| Display: | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\figures\but.jpg |

Properties:

1. CausesValidation – Gets or sets a value indicating whether validation is performed when the Button control is clicked
2. OnClientClick – Gets or sets the client-side script that executes when a Button control's Click event is raised.
3. PostBackUrl – Gets or sets the URL of the page to post to from the current page when the Button control is clicked. Text – Gets or sets the text caption displayed in the Button control.

Events:

Click – Occurs when the Button control is clicked. Calls the *Click* event handler.

Note: By default, when a button is pressed, it always does a postback to the server, to the page the button is on. If the *PostBackUrl* is set, then the page does a postback to the page specified in this property. If *OnClientClick* is set to a JavaScript function, then the page will run the function and if the function returns *True* then it will post back to the server before posting back. If the function returns *False,* the page will not postback to the server.

1. Control: [Label](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.label.aspx)

Properties:

Text – Gets or set the text caption that is displayed.

1. Control: [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx)

|  |  |
| --- | --- |
| Display: | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\figures\lab.jpg |

Properties:

1. ReadOnly – Gets or sets a value indicating whether the contents of the TextBox control can be changed.
2. Text – Gets or sets the text content of the TextBox control.
3. TextMode – Gets or sets the behavior mode (TextBoxMode.SingleLine, .MultiLine, .Password, and others)
4. Control: [Panel](https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.panel?view=netframework-4.7.2) – Represents a control that acts as a container for other controls
5. Control: [Calendar](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.calendar.aspx)

|  |  |
| --- | --- |
| Display: |  |

Properties:

1. SelectedDate – Usually used to gets the selected date, returning a *DateTime* object. Can also be used to set the selected date.
2. SelectedDates – Usually used to get the selected dates, returning a *SelectedDatesCollection* of *DateTime* objects.
3. SelectionMode – The calendar has three selection modes: CalendarSelectionMode.Day, .DayWeek, .DayWeekMonth. It does not have a way to select an arbitrary range of dates.

Events:

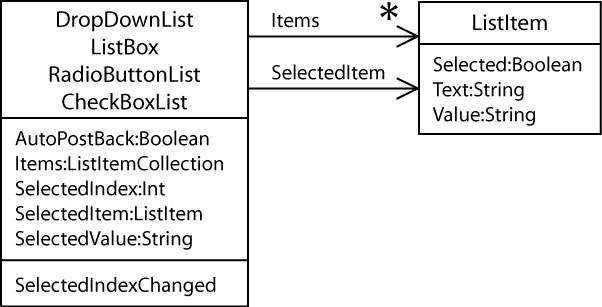
SelectionChanged – Occurs when a day (or week or month) is selected.

1. The four controls listed below have different appearances but internally operate similarly (as we will see shortly)

|  |  |  |
| --- | --- | --- |
| **Control** | **Display** | **Notes** |
| [DropDownList](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.dropdownlist.aspx): | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\dd2.jpg | Only one item may be selected |
| [ListBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\figures\lb.jpg | One item may be selected if *SelectionMode* is *Single.* Multiple items can be selected if *SelectionMode* is *Multiple*. |
| [CheckBoxList](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.checkboxlist.aspx) | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\ck.jpg | One or more items may be selected |
| [RadioButtonList](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.radiobuttonlist.aspx) | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\rb.jpg | Only one item may be selected |

# Introduction to List Controls

1. The *DropDownList (ddl), ListBox (lst), RadioButtonList (rbl), and CheckBoxList (cbl)* controls have the following properties and events in common:



1. The *Items* property is a *ListItemColllection* which can be thought of as an array of *ListItems* which can be accessed by an index (and other ways). For instance, suppose (for any of the four controls) the collection of *ListItem’s* is:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Text | Value | Selected |
| 0 | Jones | 870333232 | False |
| 1 | Williams | 870556782 | False |
| 2 | NeSmith | 870349216 | True |
| 3 | Thomas | 870890223 | False |

Then, the following properties would have these values:

|  |  |
| --- | --- |
| **Property** | **Value** |
| SelectedIndex | 2 |
| SelectedItem | |  |  |  | | --- | --- | --- | | Text | Value | Selected | | NeSmith | 870349216 | True | |
| SelectedValue | 870349216 |

Initially, the *SelectedIndex* is set to -1 which means that nothing has been selected. Once the user chooses an item, the *SelectedIndex* is set to the index of that item.

## Example

1. Example Code. Here we assume we have a *ListBox*, but the code is identical for the other controls:
2. Loop through all the *Items*:

**foreach (ListItem li in lstPlayers.Items) {**

**msg += li.Text + ", " + li.Value + ", " + li.Selected;**

**}**

*Or*

**for (int i = 0; i < lstPlayers.Items.Count; i++) {**

**ListItem li = lstPlayers.Items[i];**

**msg += li.Text + ", " + li.Value + ", " + li.Selected;**

**}**

1. Access the *SelectedValue*:

**msg += lstPlayers.SelectedValue;**

1. Access the *SelectedItem*:

**ListItem selItem = lstPlayers.SelectedItem;**

**msg += selItem.Text + ", " + selItem.Value + ", " + selItem.Selected;**

Note:

* *SelectedItem* is *null* if nothing is selected.
* The *ListBox* and *CheckBoxList* allow for multiple selections. If multiple items are selected in a ListBox, only the first item is available in the *SelectedItem* property. In this case, you need to loop through all the items and determine which are selected.

1. Determine if anything has been selected:

**if (lstPlayers.SelectedItem != null)**

*Or*

**if( lstPlayers.SelectedIndex != -1)**

## SelectedIndexChanged Event

1. All four of these controls also fire the *SelectedIndexChanged* event and have the *AutoPostBack* property set to *false* by default. In other words, when the user makes a selection from any of these controls, the page does NOT postback automatically. What happens is that if there is an event handler defined for *SelectedIndexChanged* then the event is registered when a selection takes place. When some other control (like a Button) causes a postback, then the event handler is called.

## Most common scenario: No Event Handler

1. Most common scenario:
2. We have a DropDownList (and/or any of the other 3) and (i) *AutoPostBack* is set to *false* and (ii) no event handler is defined
3. We have a *Button* with a *Click* event handler defined.
4. The user makes a selection from the *DropDownList* and presses the *Button* which causes a post back.

The server responds by

1. Calling *Page\_Load*
2. Then calling the *Button’s* *Click* event handler which would determine which item was selected in the *DropDownList* and take some action.

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**// Called first**

**}**

**protected void btnSubmit\_Click(object sender, EventArgs e)**

**{**

**// Called next, does something with DropDownList**

**}**

## Other Scenarios

1. Another scenario:
2. We have a DropDownList (or any of the other 3) and (i) *AutoPostBack* is set to *true* and (ii) we do have an event handler defined
3. The user makes a selection from the *DropDownList* which causes a post back.

The server responds by

1. Calling *Page\_Load*
2. Then calling the *DropDownList’s* *SelectedIndexChanged* event handler.

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**// Called first**

**}**

**protected void ddlMoiveChoice\_SelectedIndexChanged(object sender, EventArgs e)**

**{**

**// Called next, does something with DropDownList**

**}**

1. Another scenario:
2. We have a DropDownList (or any of the other 3) and (i) *AutoPostBack* is set to *false* and (ii) we do have an event handler defined
3. We have a *Button* with a *Click* event handler defined.
4. The user makes a selection from the *DropDownList* and presses the *Button* which causes a post back.

The server responds by

1. Calling *Page\_Load*
2. Then calling the *DropDownList’s* *SelectedIndexChanged* event handler.
3. Then calling the *Button’s Click* event handler

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**// Called first**

**}**

**protected void ddlMoiveChoice\_SelectedIndexChanged(object sender, EventArgs e)**

**{**

**// Called second, does something with DropDownList**

**}**

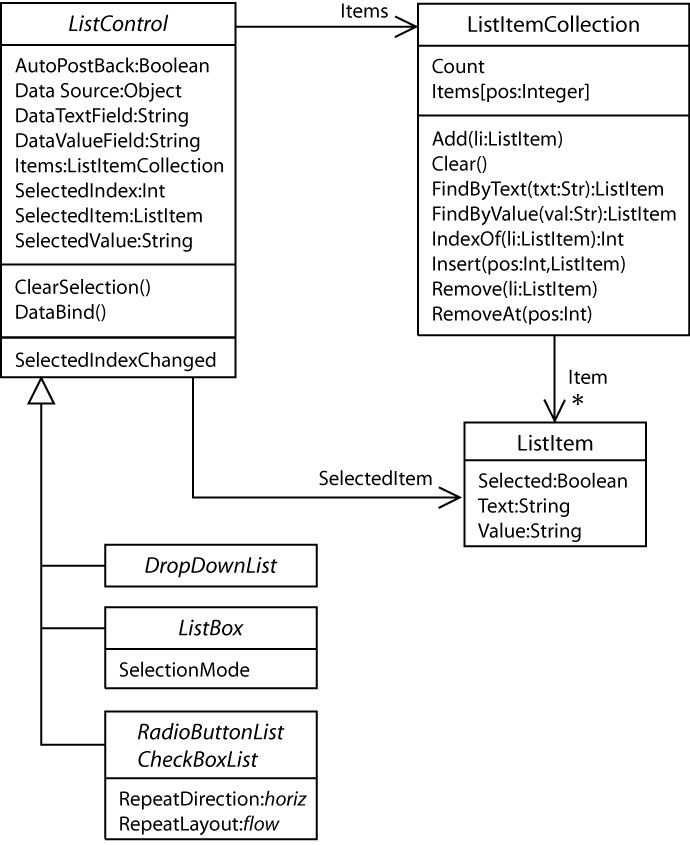
**protected void btnSubmit\_Click(object sender, EventArgs e)**

**{**

**// Called third, does something**

**}**

## A fuller description of the List Controls:



Note that *Items* is actually a *ListItemCollection* and is defined as an *indexer* which we discuss shortly.

## Value Must be Unique

1. The *value* in a DropDownList/ListBox must be unique to ensure proper operation. In real problems it is almost always a key field from a database (which is always unique). We will do this later in the course.

This is how it works:

1. Suppose two items have the same value. For example, notice below that Thomas & Williams both have value of 5.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Text | Value | Selected |
| 0 | Jones | 8 | False |
| 1 | Williams | 5 | False |
| 2 | NeSmith | 1 | False |
| 3 | Thomas | 5 | False |

1. The user selects “Thomas”. Thus, on the client-side, this is what is represented:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Text | Value | Selected |
| 0 | Jones | 8 | False |
| 1 | Williams | 5 | False |
| 2 | NeSmith | 1 | False |
| 3 | Thomas | 5 | True |

1. The *value* “5” is posted-back to the server.
2. On the server, the *ListBox* (or *DropDownList*) object is created. As it is being created, the server uses the *value* that was posted-back to search through the *ListBox* (or *DropDownList*) to find the selected item. It starts at the beginning of the list and finds the first match. Thus, on the server, internally, the control looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Text | Value | Selected |
| 0 | Jones | 8 | False |
| 1 | Williams | 5 | True |
| 2 | NeSmith | 1 | False |
| 3 | Thomas | 5 | False |

## Adding & Removing ListItems

1. Suppose you want to add a *ListItem* to a *ListBox* (or any of the other 3 list controls) and you have *TextBoxes* to allow the user to specify the *Text* and *Value.* A *Button* click event to handle this would look like this:

protected void btnAdd\_Click(object sender, EventArgs e) {

ListItem li = new ListItem();

li.Text = txtText.Text;

li.Value = txtValue.Text;

lstPlayers.Items.Add(li);

}

1. Suppose you want to remove the selected item from a single-selection *ListBox* or *DropDownList* via a button click event:

protected void btnRemoveOne\_Click(object sender, EventArgs e) {

lstPlayers2.Items.Remove(lstPlayers2.SelectedItem);

}

1. Removing selected items from a multi-select *ListBox* or *DropDownList* via a button click: An approach that works is to iterate through the items in reverse order so that when an item is deleted the renumbering only affects items that have already been processed.

protected void btnRemoveAllSelected\_Click(object sender, EventArgs e) {

for (int i = lstPlayers.Items.Count - 1; i >= 0; i--) {

if (lstPlayers.Items[i].Selected) {

lstPlayers.Items.Remove(lstPlayers.Items[i]);

}

}

}

1. Another approach for removing all selected items from a multi-select *ListBox* or *DropDownList* via a button click:

protected void btnRemoveAllSelected2\_Click(object sender, EventArgs e) {

int count = 0;

List<ListItem> selectedItems = new List<ListItem>();

if (lstPlayers.SelectedIndex != -1) {

// Get selected items

foreach (ListItem item in lstPlayers.Items) {

if (item.Selected) {

selectedItems.Add(item);

}

}

// Loop over selected items and remove

foreach (ListItem item in selectedItems) {

lstPlayers.Items.Remove(item);

count++;

}

txtMsg.Text = "" + count + " items removed";

}

else {

txtMsg.Text = "Nothing selected";

}

}

## Removing ListItems – Approaches that don’t Work

1. Removing selected items from a multi-select *ListBox* or *DropDownList* via a button click event is a bit more involved.
2. You can’t use a for-each loop as the contents of the collection cannot be modified with such a loop (same as Java). For example, this will compile, but will bomb if there is an attempt to remove an item.

protected void btnRemoveAllSelected\_Click(object sender, EventArgs e)

{

String msg = String.Empty;

foreach (ListItem li in lstPlayers.Items)

{

if (li.Selected)

{

lstPlayers.Items.Remove(li);

}

}

}

1. You can’t use an indexed for loop that starts at the beginning because it will skip the second of two consecutively selected items. For example:

protected void btnRemoveAllSelected\_Click(object sender, EventArgs e)

{

for (int i=0; i<lstPlayers.Items.Count; i++)

{

if (lstPlayers.Items[i].Selected)

{

lstPlayers.Items.Remove(lstPlayers.Items[i]);

}

}

}

Explanation – Suppose these two items (Williams & NeSmith) are selected. Consider what happens to the items as loop begins and ends for each iteration of the loop.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Beginning of Loop | End of Loop |  |
| i=0 | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | Williams | 5 | True | | 2 | NeSmith | 1 | True | | 3 | Thomas | 9 | False | | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | Williams | 5 | True | | 2 | NeSmith | 1 | True | | 3 | Thomas | 9 | False | | Item at i=0 not selected, so now deletion |
| i=1 | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | Williams | 5 | True | | 2 | NeSmith | 1 | True | | 3 | Thomas | 9 | False | | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | NeSmith | 1 | True | | 2 | Thomas | 9 | False | | Item at i=1 (Williams) is removed, then the ones following are renumbered. Notice that the item at i=2 (NeSmith) is renumbered to i=1. |
| i=2 | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | NeSmith | 1 | True | | 2 | Thomas | 9 | False | | |  |  |  |  | | --- | --- | --- | --- | | Index | Text | Value | Selected | | 0 | Jones | 8 | False | | 1 | NeSmith | 1 | True | | 2 | Thomas | 9 | False | | Thus, NeSmith was skipped. |

Appendix

1. n/a