Lab 4 – ASP.NET Basics 1b

Contents

[1 Lab Objectives 1](#_Toc94960376)

[2 Create Lab 4 Home Page 2](#_Toc94960377)

[3 Hide/Show Elements & Panels 2](#_Toc94960378)

[4 Accessing Items in a *DropDownList* 4](#_Toc94960379)

[5 Illustrate why the *Value* needs to be Unique 6](#_Toc94960380)

[6 Add Items to a *DropDownList* 7](#_Toc94960381)

[7 Remove an Item from a *DropDownList* 7](#_Toc94960382)

[8 Defining an Event Handler for a *DropDownList*, Order of Events, *AutoPostBack* 8](#_Toc94960383)

[9 Using a ListBox 9](#_Toc94960384)

[10 Using a CheckBoxList 12](#_Toc94960385)

[11 Using a RadioButtonList 12](#_Toc94960386)

[12 Using a Calendar 13](#_Toc94960387)

[13 Package Assignment for Submission 13](#_Toc94960388)

# Lab Objectives

1. Learn to use these web controls: *Panel, DropDownList, ListBox, CheckBoxList, RadioButtonList*
2. Learn what the *AutoPostBack* property does and when to set it to *true*.
3. Learn the order of how events are executed on the server.
4. Learn to hide and show controls.

To make this document easier to read, it is recommended that you turn off spell checking and grammar checking in Word:

1. Choose: File, Option, Proofing
2. At the very bottom, check: “Hide spelling errors…” and “Hide grammar errors…”

# Create Lab 4 Home Page

1. Create your *lab04* project (solution folder must be named *lab04\_lastName*).
2. Add a web form named, *default.*
3. Add the HTML title: “Lab 4 – FirstName LastName”, substituting your name.
4. Write a level 2 header, inside the *<div>* tags, that reads: “Lab 4 – FirstName LastName”
5. Verify that your page displays correctly

# Hide/Show Elements & Panels

1. Do the following:
2. Add a button to your page, give it the name (ID): *btnHideShow* and set the *Text* property to “Hide”.
3. Add a textbox to your page, give it the name: *txtInput.*
4. Create an event-handler for the button (double-click it in *Design* view).
5. Write the code to hide and show the textbox and change the text on the button. Your event handler will look like this:

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

{

if(txtInput.Visible)

{

txtInput.Visible = false;

btnHideShow.Text = "Show";

}

else

{

txtInput.Visible = true;

btnHideShow.Text = "Hide";

}

}

1. Run your page and verify that it works correctly. Note: Right-click, View in Browser only works on as *.aspx* page, not on the code-behind file.
2. Do the following:
3. Add another button to your page below the first button (do not copy/paste the earlier one), give it the name (ID): *btnHideShow2* and set the *Text* property to “Hide”.
4. Add a *Panel* to your page below the button. We will just use the default name, *Panel1*. Set the *BorderSyle* property to “Solid”.
5. Add a label and textbox inside the panel. No need to set any properties. Make sure the label and textbox are inside the panel. You can see this in Design mode, but you can also verify in Source mode
6. Create an event-handler for the new button (double-click it in *Design* view).
7. Write the code to hide and show the panel changing the text on the button. Your event handler will look like this:

protected void btnHideShow2\_Click(object sender, EventArgs e)

{

if (Panel1.Visible)

{

Panel1.Visible = false;

btnHideShow2.Text = "Show";

}

else

{

Panel1.Visible = true;

btnHideShow2.Text = "Hide";

}

}

Note that this code will hide and show the panel (and anything in it).

1. Run your page and verify that it works correctly.
2. The panel is a bit “severe”, spans the width of the page, and hugs the components inside tightly. You can improve that with a bit of jQuery. Add this to the head section:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script>

$(document).ready(function () {

// Get width of label and textbox

var lblWidth = $('#Label1').width();

var txtWidth = $('#TextBox1').width();

// Set width of panel to 20% larger than the sum of components width

$('#Panel1').width(lblWidth + txtWidth \* 1.20);

// Set padding around panel

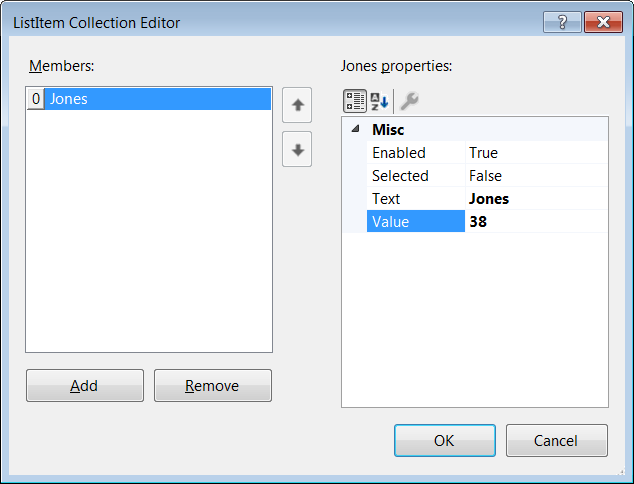
$('#Panel1').css({ 'padding': '10px' });

});

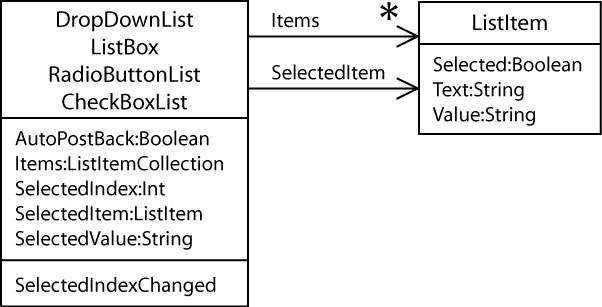
</script>

1. Run your page and observe the difference in formatting. Note: I won’t be requiring you to do formatting like this on your assignments.

# Accessing Items in a *DropDownList*

1. Add a web form named, *page2.*
2. Link *page2* to *default* and visa-versa. Do the following:
3. Open *default* (not *page2*) in *Design* view. Drag a *HyperLink* control from the ToolBox on the page, just below your name at the top.
4. Set the *Text* property to: “Page 2”
5. Set the *NaviageUrl* property to: “page2.aspx” (Note: if you navigate to the page instead of typing it in, it will show this property as: “~/page2.aspx”. The tilde means the “home directory”, which is the folder we are in. Thus, the tilde is not necessary.
6. Open *page2* in *Design* view and create a *HyperLink* back to *default.aspx.* Set the *Text* property to: “Back”.
7. Run either page and verify that you can navigate back and forth between the two pages.
8. Do the following on *page2*:
9. Add a *Button*, set the name to *btnDisplayAll,* and set *Text* to: “Display All”.
10. Beside the button, add a *DropDownList,* set the name to *ddlNames*.
11. Find the *Items* property in the Properties window and expand the ellipsis.
12. The *ListItem Collection Editor* is displayed. Press, *Add* in the lower-left and type in the *Text: “Jones”* and the *Value: “38”.*
13. Add several more entries (*ListItems*). Use the ones shown below so that I can refer to these later without confusion.

|  |  |
| --- | --- |
| Text | Value |
| Able | 22 |
| Mackey | 47 |
| Benton | 11 |

1. Run your page and verify that you can choose a name from the drop down.
2. (Read, no action required) Consider the class diagram on the right:

* A *DropDownList* has a list (named, *Items*) of *ListItem*.
* Each *ListItem* has a *Text* and *Value* (and two others) property.
* The *Text* property is what is displayed when the page is executed.
* The *Value* is what uniquely identifies the *ListItem*.
* The *Value* should be unique to ensure proper operation. The *Value* is almost always a key field from a database (which is always unique). The idea is that when the user selects an item from the drop down, then the code can retrieve the associated *Value* and then use that to look up additional information in a database. For example, we may have a drop down displaying a list of employee names (*Text* property) and the associated *Value* is the (unique) employee Id. Then, when a name is selected, we use the employee Id to look up and display their pay records. We consider the integration with a database later in the semester.
* We will consider the other properties shown for the *DropDownList* as we move along.
* All of the bullets above are true not only for a *DropDownList,* but also for a *ListBox, RadioButtonList,* and a *CheckBoxList*.

1. Do the following:
2. Add a *TextBox* below the button and drop down. Set the name to, *txtMessage.*
3. Set the *TextMode* property to *MultiLine*
4. Stretch the *TextBox* a bit larger. We will be outputting text there.
5. Do the following:
6. Create an event handler for the button (double-click it)
7. Type or add this code to the event handler

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

{

string msg = string.Empty;

foreach (ListItem li in ddlNames.Items)

{

msg += "Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected

+ Environment.NewLine;

}

txtMessage.Text = msg;

}

Note:

* *string* is an alias for *String.* You can use either.
* C# provides a *foreach* loop, which in this case would be identical to this in Java:

for(ListItem li : ddlNames.Items)

* The code simply loops through all the *ListItem* objects in *ddlNames.Items* list and collects various properties in a message which is displayed in the multi-line textbox.
* A *ListItem* has a *Selected* property that tells whether it has been selected or not.

1. Run your page and verify that the button works properly. Select a name from the dropdown and press the button again. Notice that the name you selected has the *Selected* property set to *true.*
2. Do the following:
3. Add a *Button* to *page2* beside the other button. Set the name to *btnDisplaySelected,* and set *Text* to: “Display Selected”.
4. Create an event handler for the button (double-click it)
5. Type or add this code to the event handler

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

{

ListItem li = ddlNames.SelectedItem;

string msg = "Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected;

txtMessage.Text = msg;

}

Note:

* This event handler only displays the selected *ListItem*.
* The *DropDownList* has a *SelectedItem* property which is the *ListItem* object that has been selected. Thus, this is a convenience method as you could have looped through all the *ListItems* to find which one was selected. In other words, this code would produce identical results:

string msg = string.Empty;

foreach (ListItem li in ddlNames.Items)

{

if(li.Selected)

{

msg += "Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected

+ Environment.NewLine;

break;

}

}

txtMessage.Text = msg;

1. Run your page and alternately select a name, press a button, and then press the other button; repeat. Do this to understand how the *DropDownList* remembers which *ListItem* is selected.

# Illustrate why the *Value* needs to be Unique

1. Do the following:
2. Select the *DropDownList* and expand the *Items* property.
3. Change the last name, *Benton,* to have a *Value* of 38. Notice that the first item also has a *Value* of 38.
4. Run the page, select the last name, *Benton* and then press, *Display Selected*
5. You’ll notice that *Jones* is displayed. The internal event (inside the drop down) that marks the item as selected finds the firstitem with a matching value. That probably seems a little strange. However, it does illustrate that the *Value* property needs to be unique.
6. Change the *Value* for *Benton* back to the value it was before, or something unique, say 99.

# Add Items to a *DropDownList*

1. Do the following:
2. Add a *Button* below the other buttons, and above the *TextBox.* Set the name to *btnAdd* and the *Text* to: “Add”.
3. Add a *Label* beside the button. Set the *Text* to: “Text”.
4. Add a *TextBox* beside the label. Set the name to *txtText.*
5. Add a *Label* beside the text box. Set the *Text* to: “Value”.
6. Add a *TextBox* beside the label. Set the name to *txtValue.*
7. Create an event handler for *btnAdd* (double-click)
8. Type or add the code below to the event handler, notice that I have changed to Java style “{“ on the end of line, so probably copy and replace the whole event handler:

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

if(!String.IsNullOrEmpty(txtText.Text) && !String.IsNullOrEmpty(txtValue.Text)) {

string text = txtText.Text;

string value = txtValue.Text;

ListItem li = new ListItem(text, value);

ddlNames.Items.Add(li);

txtMessage.Text = "ListItem added to dropdown";

txtText.Text = String.Empty;

txtValue.Text = String.Empty;

}

else {

txtMessage.Text = "Must supply a value for Text & Value";

}

}

1. Study the code above carefully. Run the page and experiement.

# Remove an Item from a *DropDownList*

1. Do the following:
2. Add a *Button* beside the “Display Selected” button. Set the name to *btnRemoveSelected* and the *Text* to: “Remove Selected”.
3. Create an event handler for the button and add or type this code:

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

if(ddlNames.SelectedItem != null) {

ListItem li = ddlNames.SelectedItem;

ddlNames.Items.Remove(li);

txtMessage.Text = "ListItem removed: " + li.Text + ", " + li.Value;

}

}

1. Run the page. Select a name and then choose: *Remove Selected*. Verify that the item was removed from the drop down. and verify that you can add an item to the drop down and when selected it works properly.

# Defining an Event Handler for a *DropDownList*, Order of Events, *AutoPostBack*

1. Do the following:
2. Note that we have not defined an event handler for the *DropDownList*. For many applications you do not need one. For many applications you will have a *Button* that does something with the item selected. Thus, the button is the event handler.
3. Next, we will create an event handler for the drop down so that we can observe some behaviors that are not obvious.
4. Create an event handler for the drop down by double clicking it. It will create this event handler:

protected void ddlNames\_SelectedIndexChanged(object sender, EventArgs e)

{

}

Notice that the name of the event is *SelectedIndexChanged* where as the *Buttons* have a *Click* event. Thus, *SelectedIndexChanged* is called when an item has been selected. However, there is a catch…

1. Add this code to the event handler:

ListItem li = ddlNames.SelectedItem;

string msg = Environment.NewLine + "In SelectedIndexChanged event" + Environment.NewLine

+ " Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected;

txtMessage.Text += msg;

Notice the “+=” in the last line.

1. Replace the code in the *Display Selected* event handler (*btnDisplaySelected\_Click*) with the code below.

ListItem li = ddlNames.SelectedItem;

string msg = Environment.NewLine + "In Click event" + Environment.NewLine

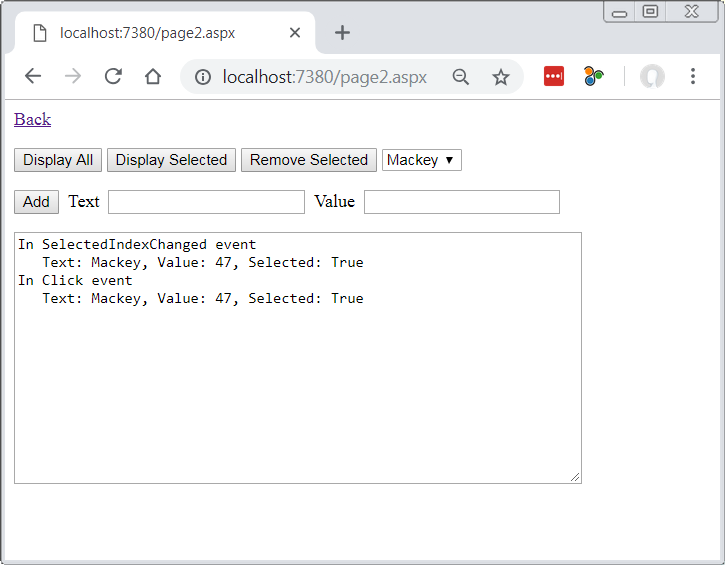
+ " Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected;

txtMessage.Text += msg;

Notice the “+=” in the last line.

1. Run the page and do the following:

* Select a name from the drop down and observe that nothing happens.
* Press *Display Selected.* The output should look as shown on the right. Notice that both events occurred.

1. (Read, no action required)

* When you selected a name from the drop down, a *SelectedIndexChanged* event was *registered* but not *executed.*
* When you pressed the *Display Selected* button, the *Click* event was registered *and* both events were executed in the order they were registered.
* The button not only registered an event when pressed, it also caused a *postback.* A *postback* means that the page is sent back to the server, where all registered events are run, and then sent back to the browser. A *Button* always causes a *postback*.
* A *DropDownList* has an *AutoPostBack* property that is set to *False* by default. We will change this next. This will force the page to postback to the server whenever a name is selected.

1. Do the following:
2. Select the drop down and set its *AutoPostBack* property to *True.*
3. Run the page and do the following:

* Select a name from the drop down and observe that the *SelectedIndexChanged* event is executed immediately (because the when you selected a name it caused a postback).
* Continue to select names and also press *Display Selected.*

1. Select the drop down and set its *AutoPostBack* property to *False.*

# Using a ListBox

1. Add a web form named, *page3* and do the following:
2. Open *default* (not *page3*) in *Design* view. Drag a *HyperLink* control from the ToolBox on the page, just below your name at the top, beside the *Page 2* link.
3. Set the *Text* property to: “Page 3”
4. Set the *NaviageUrl* property to: “page3.aspx”
5. Open *page3* in *Design* view and create a *HyperLink* back to *default.aspx.* Set the *Text* property to: “Back”.
6. Run either page and verify that you can navigate back and forth between the two pages.
7. Add a button with name *btnDisplay* and *Text* set to “Display”.
8. Add a *ListBox* and
9. Set the name to *lbxNames*
10. Set the *SelectMode* property to *Multiple*. This control allows multiple items to be selected.
11. Expand the *Items* collection and add 5 or so names making sure the values are unique. I suggest using numbers.
12. Notice that the *AutoPostBack* property is *False*. Leave it that way.
13. Add a *TextBox*, set the name to *txtMessage*, set *TextMode* to “MultiLine”, and stretch it larger.
14. Create an event handler for the button and supply this code.

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

{

string msg = string.Empty;

foreach (ListItem li in lbxNames.Items)

{

if(li.Selected)

{

msg += "Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected

+ Environment.NewLine;

}

}

txtMessage.Text = msg;

}

1. Run your page and verify.
2. Add a button between the *Display* button and the *ListBox*. Set the name to *btnRemove* and *Text* set to “Remove Selected”.
3. Create an event handler for the button and supply this code.

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

string msg = "Items removed:" + Environment.NewLine;

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

ListItem li = lbxNames.Items[i];

if (li.Selected) {

lbxNames.Items.Remove(li);

msg += "Text: " + li.Text + Environment.NewLine;

}

}

txtMessage.Text = msg;

}

Note:

* We are using an indexed loop to iterate over the *Items*. The reason is the same as Java, you can’t modify a collection when using a *foreach* loop.
* The *Items* property (the list of *ListItems*) has a *Count* property, which is equivalent to *size()* in Java
* We access the current *ListItem* with a new syntax:

ListItem li = lbxNames.Items[i];

The notation looks like Java’s array notation and in some respects it is. However, *Items* is actually a structure called an *indexer* (Java does not have this). An *indexer* is essentially a structure somewhere between an array and a list. We will consider this further in class.

* The code above compiles, but is not correct! We see this next.

1. Do the following:
2. Run your page.
3. Select the first two names.
4. Press the *Remove Selected* button.
5. Verify that only the first item was removed.
6. Close your page and run again.
7. Select the first and third names.
8. Press the *Remove Selected* button.
9. Verify that both names were removed.

Can you figure out why this code performed incorrectly when the first two items were selected? If you took CS 1302 from me, we illustrated this when we considered removing items from an *ArrayList* using an indexed loop.

Here’s the problem: all the *ListItems* have an index: 0, 1, …. So, when the loop is on *i=0,* that item is selected and so it is removed properly. However, once it is removed, all the other items are re-indexed. Thus, the second item, which had *index=1,* changes to *index=0, etc.* In the meantime, the loop increments *i* to 1. Thus, we skipped over the previously second item because it is now first. We fix this next, by iterating over the *ListItems* in reverse order.

1. Replace the code in the event handler with this code:

string msg = "Items removed:" + Environment.NewLine;

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

{

ListItem li = lbxNames.Items[i];

if (li.Selected)

{

lbxNames.Items.Remove(li);

msg += "Text: " + li.Text + Environment.NewLine;

}

}

txtMessage.Text = msg;

By iterating in reverse order, when we remove a *ListItem,* the only items that are re-indexed are ones that we have previously processed in the loop. Thus, we don’t skip any selected items.

1. Run your page, experiment, and verify that the code is running correctly. To experiment thoroughly, you should rerun your page from VS several times.

# Using a CheckBoxList

1. Add a web form named, *page4* and link this page to *default* and link *default* to this page.
2. Add a *CheckBoxList* to *page4* and do the following:
3. Set the name to *cblTeams*
4. Set the *RepeatDirection* property to *Horizontal*.
5. Expand the *Items* collection and add 5 or so teams. Make the *Text* the school name (*e.g.* VSU) and the *Value* the sports name (*e.g.* Blazers).
6. Notice that the *AutoPostBack* property is *False*. Leave it that way.
7. Add a button with name *btnDisplay* and *Text* set to “Display”.
8. Add a *TextBox*, set the name to *txtMessage*, set *TextMode* to “MultiLine”, and stretch it larger.
9. Create an event handler for the button and supply this code.

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

{

string msg = "Your favorite teams are:" + Environment.NewLine;

foreach (ListItem li in cblTeams.Items)

{

if (li.Selected)

{

msg += "Text: " + li.Text + ", "

+ "Value: " + li.Value + ", "

+ "Selected: " + li.Selected

+ Environment.NewLine;

}

}

txtMessage.Text = msg;

}

1. Run your page and verify.

# Using a RadioButtonList

1. Add a web form named, *page5* and link this page to *default* and link *default* to this page. **Warning: do not copy/paste *page4* in the SE. It will not work correctly.**
2. Make an example similar to the previous stage (CheckBoxList) except use a *RadioButtonList*. Hint: you do not have to loop through all the radio buttons to see which one is selected. In a *RadioButtonList,* only one item can be selected. Thus, in your event handler, simply use the *SelectedItem* property of the *RadioButtonList.*

# Using a Calendar

1. Add a web form named, *page6* and link this page to *default* and link *default* to this page.
2. Add a button with name *btnDisplay* and *Text* set to “Display”.
3. Add a *Calendar* with name *cal* and verify that *SelectionMode* is set to “Day”.
4. Add a *TextBox*, set the name to *txtMessage*, set *TextMode* to “MultiLine”, and stretch it larger.
5. Create an event handler for the button and supply this code:

DateTime date = cal.SelectedDate;

string msg = "Date properties" + Environment.NewLine;

msg += "Short date string:" + date.ToShortDateString() + Environment.NewLine;

msg += "Day:" + date.Day;

msg += ", Month:" + date.Month;

msg += ", Year:" + date.Year + Environment.NewLine;

msg += "Day of week:" + date.DayOfWeek + Environment.NewLine;

msg += "Day of year:" + date.DayOfYear + Environment.NewLine;

// Illustrates the AddDays method

DateTime date2 = date.AddDays(3);

msg += "3 days from now:" + date2.ToShortDateString() + Environment.NewLine;

txtMessage.Text = msg;

1. Run, select a date, press the button and observe the output.

# Package Assignment for Submission

1. Close VS and zip your *lab04\_lastName* solution folder and submit on Blazeview in the *Lab 04* dropbox.

If you need further directions, follow step 10 from Lab 1, exactly, substituting *lab04* for *lab01*.

|  |
| --- |
| To avoid a deduction of points, verify the following:   1. The name of your zip file is: *lab04\_lastName* (your last name, of course). 2. Inside the zip file, verify that your *lab04\_lastName* solution folder is indeed named: *lab03\_lastName* 3. Inside your *lab04\_lastName* solution folder there should be:  * A *lab04\_lastName.sln* file * The page(s) you developed * Some other files & folders. |