**Lab 7 – CS 3340**

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…”

**Lab Objectives**

1. Implement Ajax using JavaScript
2. Implement Ajax using jQuery
3. Implement Ajax using .Net Server Controls

**Lab Organization**

There are 9 stages to complete Lab 7.

|  |  |
| --- | --- |
| Stage | Description |
| 0 | Synchronous & Asynchronous Communication with Server |
| 1 | AJAX with JavaScript |
| 2 | AJAX with jQuery |
| 3 | AJAX with .Net Server Controls |
| 4 | Handle a Slow Ajax Call |
| 5 | Add PostBack Behavior |
| 6 | Add a Trigger |
| 7 | Use the Timer Control |
| 8 | Package Assignment for Submission |

**Stage 0 – Synchronous & Asynchronous Communication with Server**

(Read, no action required)

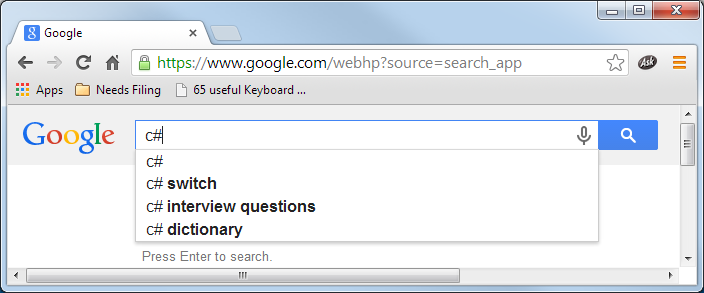
1. Up to now, we have used a *synchronous* model for communication with the server:

|  |  |
| --- | --- |
| **Client** | **Server** |
| User interacts with page and causes a postback |  |
| *User waits for response from Server* | Processes page and sends to client |
| Entire page refreshed |  |
| User interacts… |  |

1. *AJAX* is a technique that allows a page to communicate with the server *asynchronously*. What this means is that when an AJAX request is made to the server, the page continues to function. Eventually, the AJAX response is received from the server and the page (partially) updates.

|  |  |
| --- | --- |
| **Client** | **Server** |
| User interacts with page and causes a partial postback |  |
| User continues to interact with page | Processes page and sends to client |
| Page is partially refreshed |  |
| User continues to interact… |  |

In other words, there is no “waiting”, at least in theory.

1. *AJAX* is a technique that allows a page to communicate with the server *asynchronously*. It is used in most web applications. An example is Google’s suggestion list that appears as you type in the search box. As you type, in the background AJAX is used to send the character(s) you are typing to the server. The server searches for topics that match and sends the results back to the browser which renders it. All the while, the user can keep typing.
2. When AJAX is **not** being used, the tab will flicker when a postback occurs. For example, visit VSU’s CS page and then choose from one of the menus on the right. You’ll see the flicker as the page reloads:

<https://www.valdosta.edu/cset/>

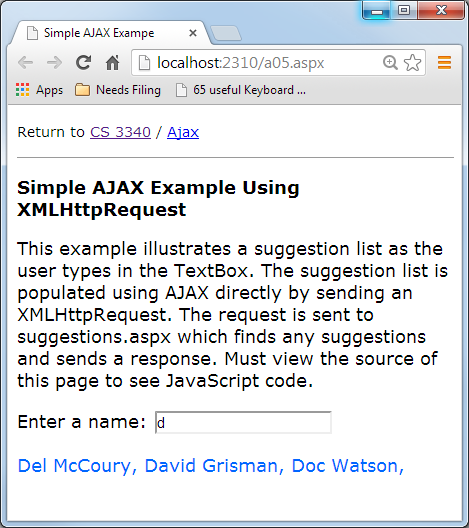
When AJAX is being used, the page will (partially) refresh but the tab will not flicker. For example, visit this page and choose one of these: Overview, Majors, Minors, Courses:

<https://catalog.valdosta.edu/undergraduate/academic-programs/sciences-mathematics/computer-science-engineering/>

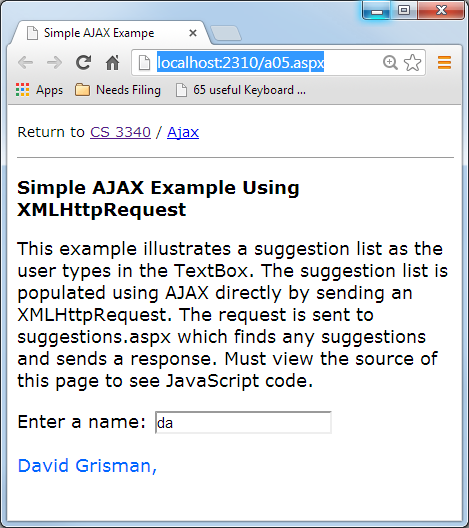
1. *AJAX* is an acronym for *Asynchronous JavaScript and XML*. It is not a programming language, but rather a technique that uses existing technologies (JavaScript and XML).

**Stage 1 – AJAX with JavaScript**

1. (Read, no action required). We will build a page that gives users “suggestions” as they type in a name. For example, when the user types, “d” the list of suggestions will be as shown in blue below:



Then, when the user types a second letter, “a”, then the suggestion list is:



1. Create your *lab07\_lastName* project.
2. Create a web form named *Suggestions.aspx* and do the following:
3. Add this method to the code-behind file (feel free to change and/or add to the list of musicians):

void GetSuggestions() {

// Get the request query

string strQuery = Request.QueryString["q"].ToString();

//Create the string to be sent back in the response.

string strSuggestions = "";

//An arbitrary array of names that are used as suggestions.

string[] arrStrNames = new string[]{ "Abbie Hoffman", "Bob Weir", "Bob Dylan", "Bill Monroe", "BB King", "Cat Stevens", "Carly Simon",

"Del McCoury", "David Grisman", "Doc Watson", "Earl Scruggs", "Frank Zappa", "Guy Clark", "Greg Allman",

"Ian Anderson", "Jack Williams", "John Doe", "Jimi Hendrix", "Janis Joplin", "Jerry Garcia", "Jimmy Rogers",

"Peter Rowan", "Pete Townsend", "Townes Van Zandt", "Verlon Thompson" };

// Loop through the names, appending matches

for (int i = 0; i < arrStrNames.GetLength(0); i++) {

// Make sure input is smaller than the current name from the list.

if (strQuery.Length <= arrStrNames[i].Length)

// See if the current name begins with the input string.

if (arrStrNames[i].ToLower().Substring(0, strQuery.Length) == strQuery.ToLower()) {

// Add name and title as an xml element.

strSuggestions += arrStrNames[i] + ", ";

}

}

// Send the response

Response.Write(strSuggestions);

// End the page lifecycle and send immediately.

Response.End();

}

1. Add a call to this method in *Page\_Load:*

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

GetSuggestions();

}

1. (Read, no action required) This page will not have a user interface. It will never be “displayed”. We are simply using it to supply a suggestion list. **Read through the comments in the code above to get a feel for what is going on.**
2. Add a web form named, *AjaxJS.aspx,* display in Source viewand do the following:
3. Add this HTML inside the *div:*

<p><b>Simple AJAX Example Using JavaScript XMLHttpRequest Directly</b></p>

<p>This example illustrates a suggestion list as the user types in the TextBox.</p>

<p>Enter a name:

<input id="txtName" onkeyup="getSuggestions(this.value);" type="text" name="txtName" />

</p>

<%--Suggestions will go in div below--%>

<div class="style1" id="nameList"></div>

Note that every time the user types a character, the *onkeyup* event is fired and we call the *getSuggestions* function passing to it all that the user has typed in.

1. Add this title: <title>Ajax via JavaScript</title>
2. Add this script in the head section

<script type="text/javascript">

var ajaxRequest; // Need to use in both functions below.

// Takes character(s) from user and sends to server

function getSuggestions(strName) {

if (strName.length == 0) { // If no data, clear suggestions.

document.getElementById("nameList").innerHTML = '';

return;

}

if (!window.XMLHttpRequest) {

alert("Browser doesn't support XMLHttpRequest object");

return;

}

// Create the (Ajax) request object.

ajaxRequest = new XMLHttpRequest();

// Set the call-back function. When the server sends its response, this

// method, displaySuggestions will be called

ajaxRequest.onreadystatechange = displaySuggesionts;

// Set the location to send the request to, "suggestions.aspx" with the

// characters the user has typed in, in the query string

ajaxRequest.open("GET", "suggestions.aspx?q=" + strName, true);

// Send request to server.

ajaxRequest.send();

}

// When server responds, displays suggestions.

function displaySuggesionts() {

// readyState of 4 or 'complete' represents that data has been returned.

if (ajaxRequest.readyState == 4 && ajaxRequest.status == 200) {

// Get the suggestions out of the request object.

var suggestions = ajaxRequest.responseText;

if (suggestions.length == 0) {

document.getElementById("nameList").innerHTML = "\*\*\*No suggestions";

return;

}

// Display suggestions.

document.getElementById("nameList").innerHTML = suggestions;

}

}

</script>

1. (Read, no action required) Read through the code and comments above. You should be able to get a general idea of what is going on. We will discuss more in class.
2. Run the page. Type a character or two that match some of the suggestions in *Suggestions.aspx*. For example, type, “j”, “ji”, “ja”, etc. You may need to use the Escape key to clear the browser’s suggestions that may pop up as you type, obscuring the display of our suggestions.

**Stage 2 – AJAX with jQuery**

Next, we will create a new page that is exactly the same as the previous page, except that we will use jQuery instead of straight JavaScript.

1. Do the following:
2. Close all open documents, select your *AjaxJS.aspx* page in the SE, copy it, select the *lab07* node, and paste. Rename it to, *AjaxJQuery.aspx*.
3. Open the Source and change the “inherits” attribute so that refers to “AjaxJQuery”:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AjaxJQuery.aspx.cs" Inherits="lab07.AjaxJQuery" %>

1. Open the code-behind file and change the name of the class to, *AjaxJQuery*:

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

1. Run the page and verify that it works properly. If not, delete it and repeat.
2. Open *AjaxJQuery* in Source mode and do the following:
3. Change the title to: <title>Ajax via jQuery</title>
4. Change the text at the top of the page so that it reads:

<p><b>Simple AJAX Example Using jQuery</b></p>

1. Remove the *onkeyup* attribute from the *input* tag so that it looks like this:

<input id="txtName" type="text" name="txtName" />

1. Delete all the JavaScript including the script tags.
2. Add the scripts below

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

<script>

function noSuggestions() {

$("#nameList").text("\*\*\*No suggestions");

}

$(document).ready(function() {

$("#txtName").keyup(function () {

var userData = $("#txtName").val();

if (userData.length == 0)

return noSuggestions();

$.ajax({

type: "GET",

url: "Suggestions.aspx/GetSuggestions",

data: { q:userData },

success: function (response) {

if (response.length == 0)

return noSuggestions();

$("#nameList").text(response);

}

});

});

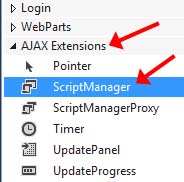
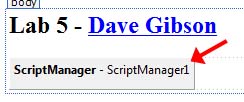
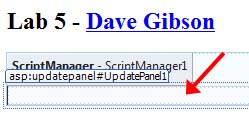
});

</script>

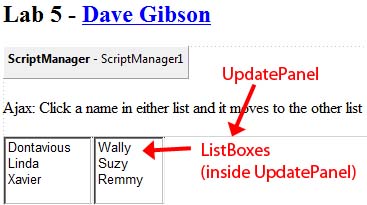
1. (Read, no action required) Read through the code (sorry, no comments!). Note the “$.ajax”, which is doing the ajax call to server and handling the response.
2. Run the page. Type a character or two that match some of the suggestions in *Suggestions.aspx*.

**Stage 3 – AJAX with .Net Server Controls**

Next, we use .Net server controls to implement Ajax.

1. Create a web form named *AjaxNet.aspx* and do the following:
2. Add a title: <title>Ajax .Net Controls</title>
3. Add a header: Simple AJAX Example Using .Net Server Controls
4. Open the page in Design mode (if not already) and then open the Toolbox and expand the Ajax Extensions group
5. Add a *ScriptManager* control to the page. At design-time, it will look as shown on the right (but with the header added in *b* above). The *ScriptManager* control is called a *component.* Components do not have a visual presence at run-time, they simply provide some background infrastructure to support other controls. This *ScriptManager* component provides the infrastructure needed to use other Ajax controls.
6. Add an *UpdatePanel* (found in the Ajax Extensions group). At design-time, it will appear as shown on the right. Note the following:

* An *UpdatePanel* is also a component and is not visible at runtime.
* We will put the controls in the *UpdatePanel* which will make them *Ajaxified*.
* This is the region where *partial-page updates* occur.

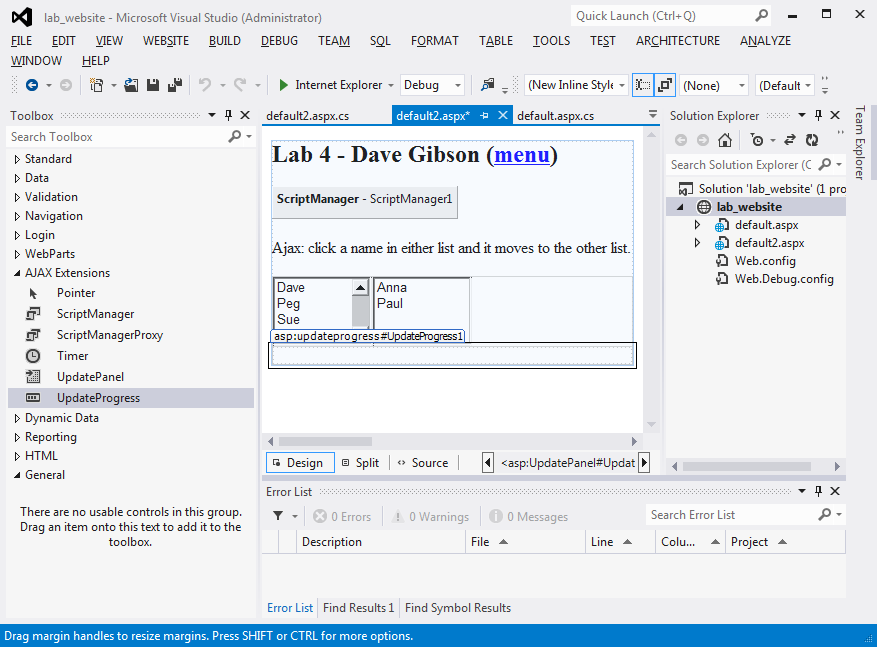
1. Design your page so that it looks as shown on the right.
2. Add the text shown above the two *ListBoxes*. It does not need to be inside the UpdatePanel.
3. Add two ListBoxes inside the UpdatePanel. Give them a meaningful ID, add some names, set *AutoPostBack* to true for both.
4. View the Source of your page and make sure the *ListBoxes* are inside the *UpdatePanel* (and inside the *ContentTemplate*) as shown on the right.
5. Add event handlers so that when the user selects a name in either list, it will move the name to the other list. Write the code to make this happen. Hint – Algorithm for listbox1 (similar for listbox2):
6. Get selected item from listbox1
7. Remove it from listbox1
8. Add it to listbox2
9. Set the selected index to -1 for listbox2 (This is important: otherwise, the item moved will be selected in lbx2 which will cause problems when you select another name.)

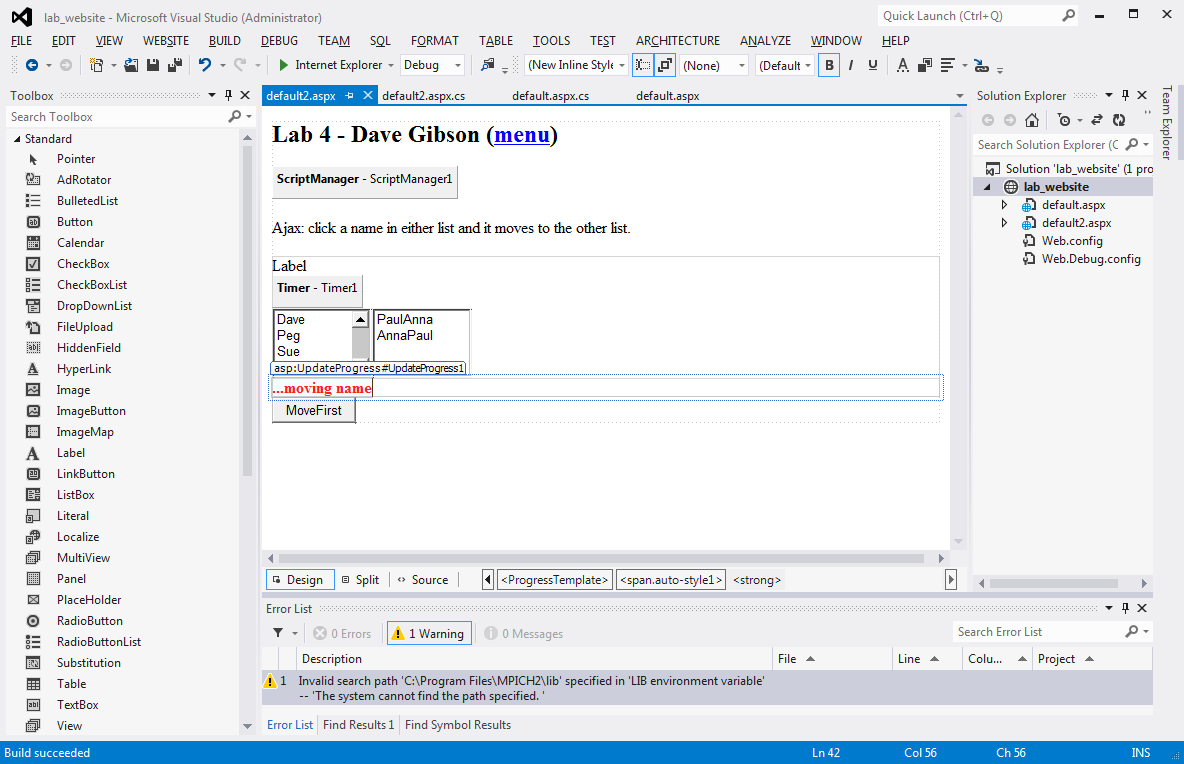
Challenge (optional): Can you do this with just one event handler? Hint: use the *sender* argument to determine which ListBox triggered the event.

1. Run and Test. You should not see the tab flicker when a name is selected. This tells us that Ajax is being used to communicate with the server.

**Stage 4 – Handle a Slow Ajax Call**

We use the Ajax *UpdateProgress* control to handle the situation where an Ajax calls takes a “long” time. This control just displays a message of your choice after the call has taken more than a certain number of seconds, which you specify.

1. Do the following:
2. Add an *UpdateProgress* control to the page (found in the Ajax Extentions group). It will appear as shown on right (ignore the “Lab 4” title). It does not have to be in the UpdatePanel



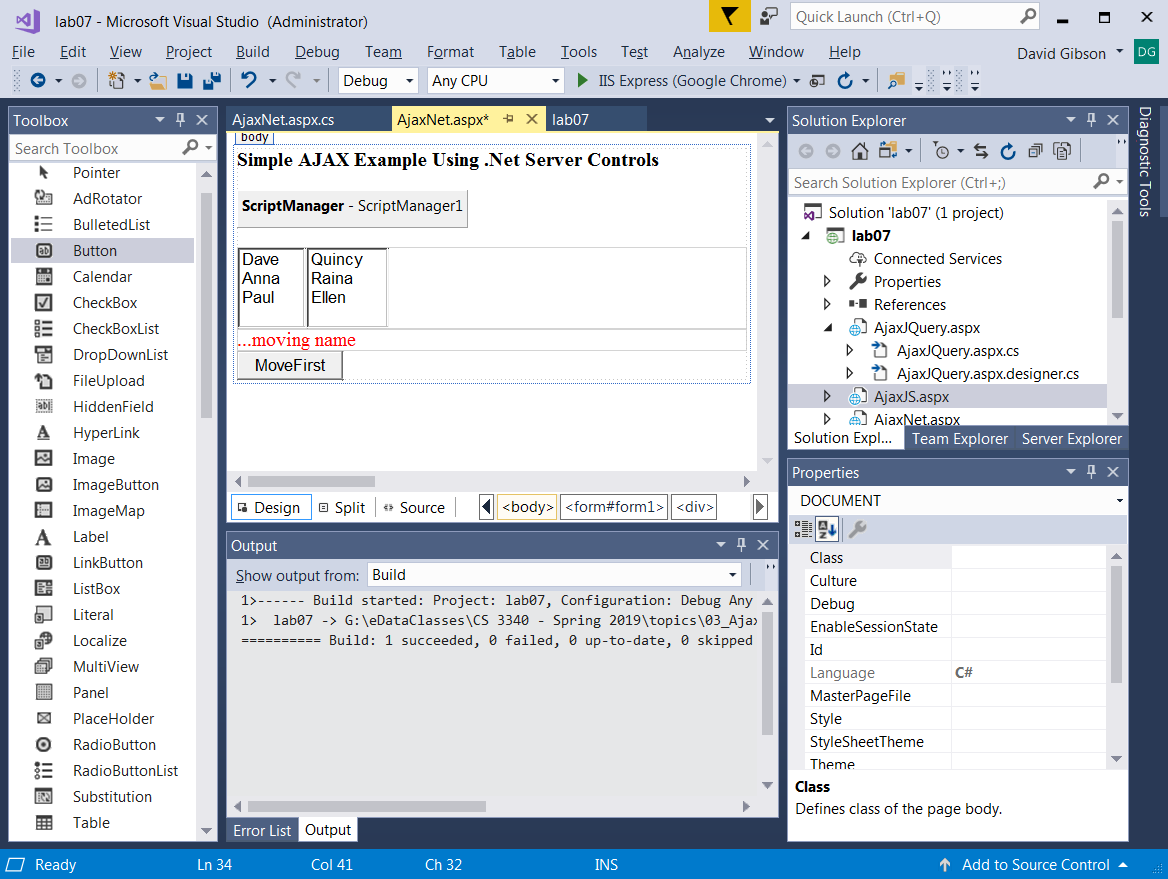
1. Add a *Label* with the text as shown in the UpdateProgress panel as shown on the right. (To make the text red, select the *Label* and set the *ForeColor*).
2. Select the UpdatePanel (click the tag at the top of the region). Note that the *DisplayAfter* property is set to 500 (0.5 seconds). This means that this message will be displayed if the Ajax call takes longer than 0.5 seconds.
3. Add an artificial delay in the method(s) that move the name. Add this line at the top of both your event handlers:

**System.Threading.Thread.Sleep(3000);**

1. Run and Test. You should see the message appear when you move a name and the tab should not flicker.

**Stage 5 – Add PostBack Behavior**

Here, we will add a Button that causes the first item in the left list to be moved. And, it will accomplish this through a postback, not Ajax.

1. Do the following
2. Add a Button but make sure it is **outside the UpdatePanel**. Supply the Text shown on the right, “Move First”.
3. Open the page in Source mode and verify that the Button is outside the UpdatePanel.
4. Add a click event handler that moves the first item in the left ListBox to the right ListBox. Thus, repeatedly pressing the button will move the names over to the right one-at-a-time. Something like this:

if(lbx1.Items.Count>0) {

ListItem li = lbx1.Items[0];

lbx1.Items.Remove(li);

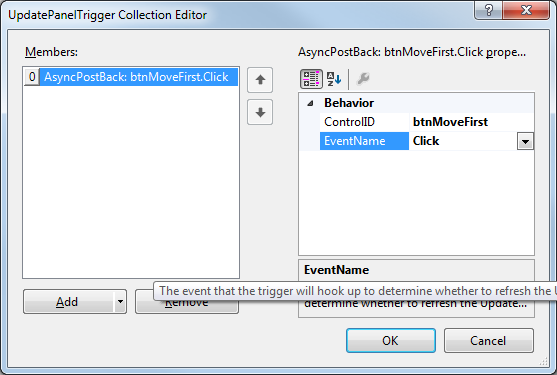
lbx2.Items.Add(li);

lbx2.SelectedIndex = -1;

}

1. Run and Test. You *should* see the tab flicker when the event is fired and there will still be a delay of 3 seconds, but the UpdateProgress from Stage 3 will not be shown. The flicker of the tab shows that the button is *not* using Ajax.

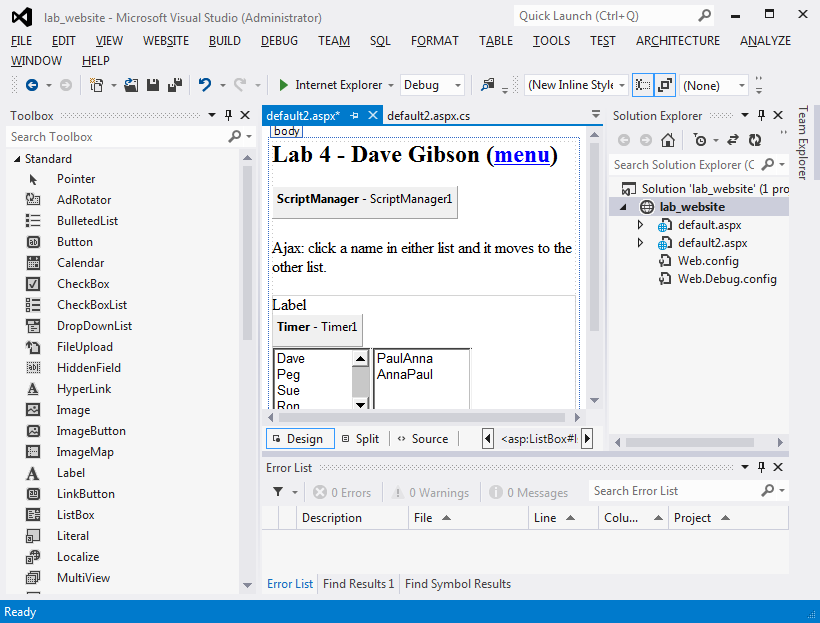
**Stage 6 – Add a Trigger**

Next, we will change the post-back button (which is outside the UpdatePanel) to use Ajax to call the click event. We do this by defining a *trigger*.

1. **Make Button use Ajax**
2. Select the UpdatePanel (click the tag at the top of the panel).
3. Choose the *Triggers* property and the dialog shown on the right will appear.
4. Choose: Add from the bottom left, and then supply the values shown on the right and click OK.
5. **Run and Test** (Ctrl+Shift B, Ctrl+F5). Now, you *should not* see the tab flicker. The Button outside the UpdatePanel is *triggering* an Ajax call to the click event-handler.

**Stage 7 – Use the Timer Control**

The *Timer* is an Ajax control that is used to call a method (the *Tick* event) on the server on a periodic basis, say every 2 seconds. Here, we will use it to display and update the time every 2 seconds.

1. **Add a Timer Control**
2. **Inside the UpdatePanel**, just above the two ListBoxes, add:
3. A Label (as shown on the right) and give it the ID: *lblTime.*
4. Add a *Timer* from the Ajax Extensions group in the Toolbox (no need to change the ID). The *Timer* is a component and is not shown at runtime.
5. Set the *Interval* property for the Timer to 2000 (which represents 2 seconds).
6. In Design view, double-click the Timer to create the *Tick* event handler. This event will be called every 2 seconds (as specified in the *Interval* from the last step). The event will update the time. Supply the code shown below:

**protected void Timer1\_Tick(object sender, EventArgs e)**

**{**

**lblTime.Text = DateTime.Now.ToLongTimeString();**

**}**

1. **Run and Test** (Ctrl+Shift B, Ctrl+F5). Make sure the time is updating and everything else works.

**Stage 8 – Package Assignment for Submission**

1. Close VS and zip your *lab07\_lastName* project folder and submit on Blazeview in the *Lab 07* dropbox.