```
System.Web.UI.WebControls.Image imgNew = new System.Web.UI.WebControls.Image();
imgNew.ImageUrl = "cellpic.png";

// Put the label and picture in the cell.
cellNew.Controls.Add(lblNew);
cellNew.Controls.Add(imgNew);

// Put the TableCell in the TableRow.
rowNew.Controls.Add(cellNew);
```

The real flexibility of the table test page is that each Table, TableRow, and TableCell is a full-featured object. If you want, you can give each cell a different border style, border color, and text color by setting the corresponding properties.

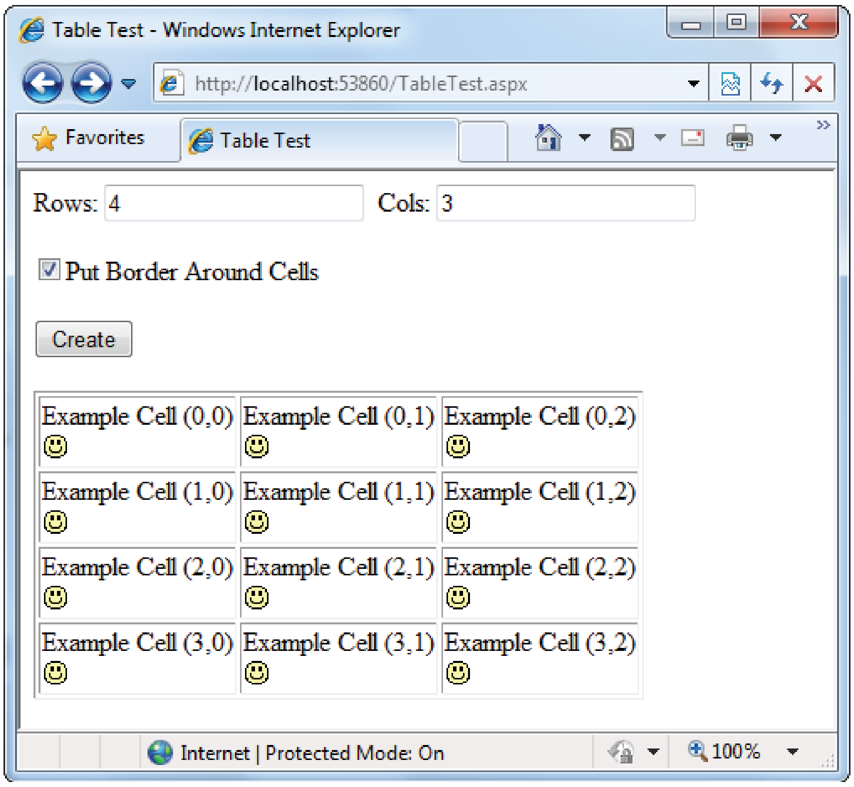
Table Test - Windows Internet Explorer
http://localhost:53860/TableTest.aspx
Favorites
Table Test
Internet | Protected Mode: On
100%

Rows: 4 Cols: 3

[✓] Put Border Around Cells

Create

Example Cell (0,0)	Example Cell (0,1)	Example Cell (0,2)
Example Cell (1,0)	Example Cell (1,1)	Example Cell (1,2)
Example Cell (2,0)	Example Cell (2,1)	Example Cell (2,2)
Example Cell (3,0)	Example Cell (3,1)	Example Cell (3,2)

Figure 6-10. *A table with contained controls*

Web Control Events and AutoPostBack

The previous chapter explained that one of the main limitations of HTML server controls is their limited set of useful events—they have exactly two. HTML controls that trigger a postback, such as buttons, raise a ServerClick event. Input controls provide a ServerChange event that doesn't actually fire until the page is posted back.

ASP.NET server controls are really an ingenious illusion. You'll recall that the code in an ASP.NET page is processed on the server. It's then sent to the user as ordinary HTML. Figure 6-11 illustrates the order of events in page processing.

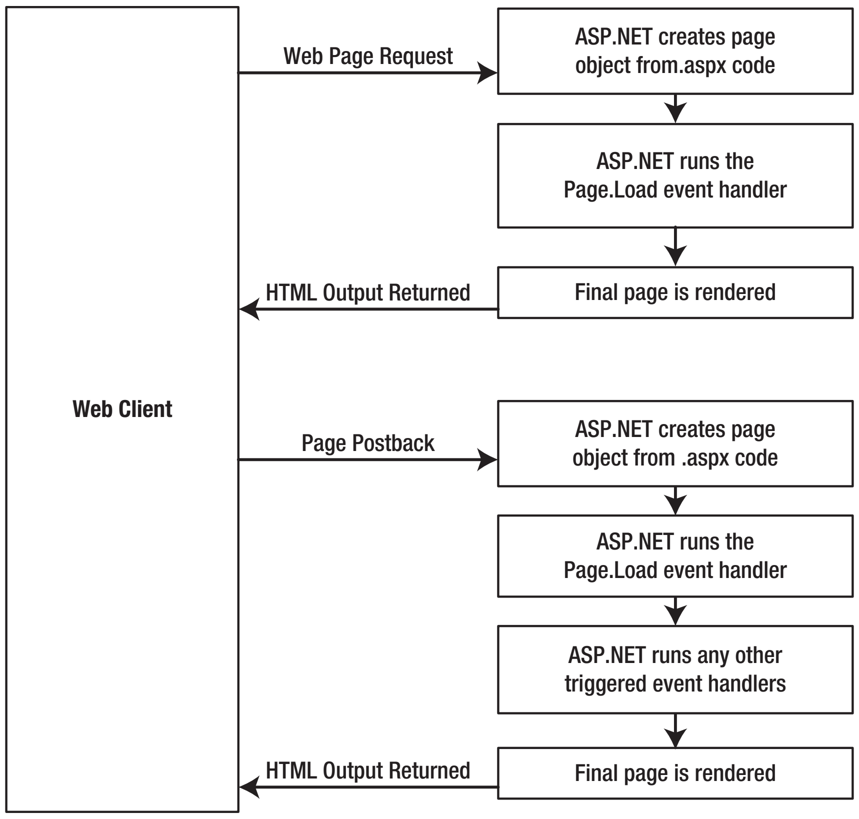
Web Client
Web Page Request
ASP.NET creates page object from.aspx code
ASP.NET runs the Page.Load event handler
HTML Output Returned
Final page is rendered
Page Postback
ASP.NET creates page object from .aspx code
ASP.NET runs the Page.Load event handler
ASP.NET runs any other triggered event handlers
HTML Output Returned
Final page is rendered

Figure 6-11. *The page-processing sequence*

This is the same in ASP.NET as it was in traditional ASP programming. The question is, how can you write server code that will react *immediately* to an event that occurs on the client?

Some events, such as the Click event of a button, do occur immediately. That's because when clicked, the button posts back the page. This is a basic convention of HTML forms. However, other actions *do* cause events but *don't* trigger a postback—for example, when the user changes the text in a text box (which triggers the TextChanged event) or chooses a new item in a list (the SelectedIndexChanged event). You might want to respond to these events, but without a postback, your code has no way to run.

ASP.NET handles this by giving you two options:

- You can wait until the next postback to react to the event. For example, imagine you want to react to the SelectedIndexChanged event in a list. If the user selects an item in a list, nothing happens immediately. However, if the user then clicks a button to post back the page, *two* events fire: Button.Click followed by ListBox.SelectedIndexChanged. And if you have several controls, it's quite possible for a single postback to result in several change events, which fire one after the other, in an undetermined order.

- You can use the *automatic postback* feature to force a control to post back the page immediately when it detects a specific user action. In this scenario, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.

The option you choose depends on the result you want. If you need to react immediately (for example, you want to update another control when a specific action takes place), you need to use automatic postbacks. On the other hand, automatic postbacks can sometimes make the page less responsive, because each postback and page refresh adds a short, but noticeable, delay and page refresh. (You'll learn how to create pages that update themselves without a noticeable page refresh when you consider ASP.NET AJAX in Chapter 25.)

All input web controls support automatic postbacks. Table 6-5 provides a basic list of web controls and their events.

Table 6-5. *Web Control Events*

Event	Web Controls That Provide It	Always Posts Back
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

If you want to capture a change event (such as TextChanged, CheckedChanged, or SelectedIndexChanged) immediately, you need to set the control's AutoPostBack property to true. This way, the page will be submitted automatically when the user interacts with the control (for example, picks a selection in the list, clicks a radio button or a check box, or changes the text in a text box and then moves to a new control).

When the page is posted back, ASP.NET will examine the page, load all the current information, and then allow your code to perform some extra processing before returning the page back to the user (see Figure 6-12). Depending on the result you want, you could have a page that has some controls that post back automatically and others that don't.

This postback system isn't ideal for all events. For example, some events that you may be familiar with from Windows programs, such as mouse movement events or key press events, aren't practical in an ASP. NET application. Resubmitting the page every time a key is pressed or the mouse is moved would make the application unbearably slow and unresponsive.

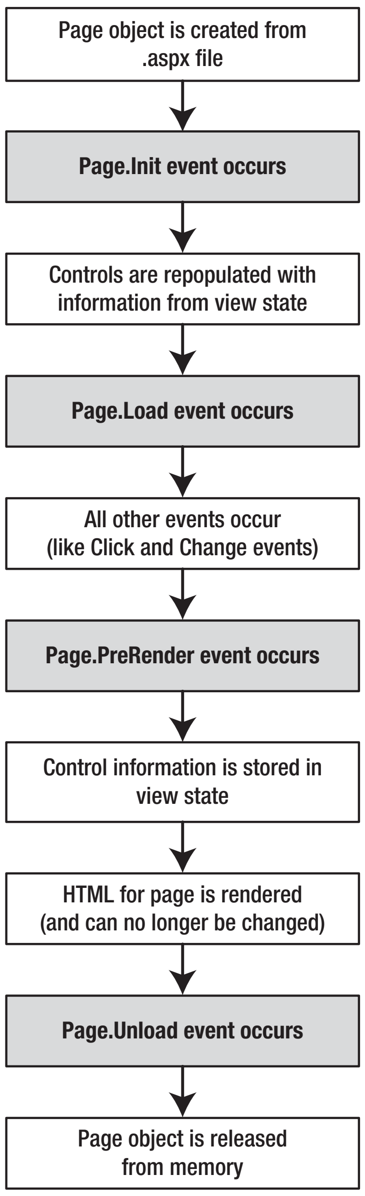
Page object is created from .aspx file
Page.Init event occurs
Controls are repopulated with information from view state
Page.Load event occurs
All other events occur (like Click and Change events)
Page.PreRender event occurs
Control information is stored in view state
HTML for page is rendered (and can no longer be changed)
Page.Unload event occurs
Page object is released from memory

Figure 6-12. *The postback processing sequence*

How Postback Events Work

Chapter 1 explained that not all types of web programming use server-side code like ASP.NET. One common example of client-side web programming is JavaScript, which uses script code that's executed by the browser. ASP.NET uses the client-side abilities of JavaScript to bridge the gap between client-side and server-side code.

(Another scripting language is VBScript, but JavaScript is the only one that works on all modern browsers, including Internet Explorer, Chrome, Firefox, Safari, and Opera.)

Here's how it works: If you create a web page that includes one or more web controls that are configured to use AutoPostBack, ASP.NET adds a special JavaScript function to the rendered HTML page. This function is named __doPostBack(). When called, it triggers a postback, sending data back to the web server.

ASP.NET also adds two hidden input fields that are used to pass information back to the server. This information consists of the ID of the control that raised the event and any additional information that might be relevant. These fields are initially empty, as shown here:

```
<input type="hidden" name="__EVENTTARGET" ID="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" ID="__EVENTARGUMENT" value="" />
```

The __doPostBack() function has the responsibility of setting these values with the appropriate information about the event and then submitting the form. The __doPostBack() function is shown here:

```
<script language="text/javascript">
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
  ...
}
</script>
```

Remember, ASP.NET generates the __doPostBack() function automatically, provided at least one control on the page uses automatic postbacks.

Finally, any control that has its AutoPostBack property set to true is connected to the __doPostBack() function by using the onclick or onchange attributes. These attributes indicate what action the browser should take in response to the client-side JavaScript events onclick and onchange.

The following example shows the tag for a list control named lstBackColor, which posts back automatically. Whenever the user changes the selection in the list, the client-side onchange event fires. The browser then calls the __doPostBack() function, which sends the page back to the server.

```
<select ID="lstBackColor" onchange="__doPostBack('lstBackColor',")"
 language="javascript">
```

In other words, ASP.NET automatically changes a client-side JavaScript event into a server-side ASP.NET event, using the __doPostBack() function as an intermediary. Figure 6-13 shows this process.

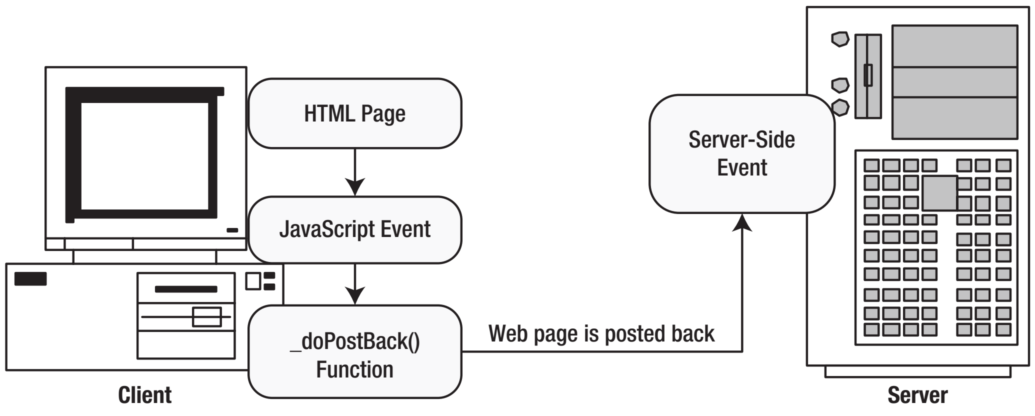

Figure 6-13. *An automatic postback*

The Page Life Cycle

To understand how web control events work, you need to have a solid understanding of the page life cycle. 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 by 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 6-14 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).