

Chapter 2

Basics of AJAX

Learning Objectives

After completing this chapter, you will be able to understand:

- *The UpdatePanel control.*
- *The UpdateProgress control.*
- *The life cycle of Page Request.*



INTRODUCTION

In this chapter, you will learn about some basic controls of AJAX such as the **UpdatePanel** control, which is the most important and dynamic control in ASP.NET AJAX. In addition to this, you will learn about a new control called the **UpdateProgress** control, and also about the life cycle of Page Request.

THE UpdatePanel CONTROL

The **UpdatePanel** control is a dynamic control that is used to update the portion of a web page asynchronously. Here, the term asynchronously indicates that you do not have to wait for the response of the server after sending a request. Generally, when you click a button on a web page, the page sends a request to the server and then immediately gets back a response from it. During this process of request and response, the entire page is reloaded. For example, when you click on the **submit** button of a form, a request is sent to the server and the entire form is reloaded. As a result, a response is generated by the server. The process of reloading the entire page is called postback. During postback, you can clearly see the flickering of the page and also the progress of reloading of the page on the Status bar of the page. While reloading, you may also hear the sound of click one or more times. Due to all this, you may feel a little interruption while working.

The **UpdatePanel** control helps you restrict the flickering of the page and implements the postback asynchronously. The **UpdatePanel** control is a container control without any user interface(UI). It is located in the **System.Web.UI** namespace.

You can find the **UpdatePanel** control in the **AJAX Extensions** section of the **Toolbox**, as shown in Figure 2-1. To place it on the page, double-click on the **UpdatePanel** control in the **AJAX Extensions** section.

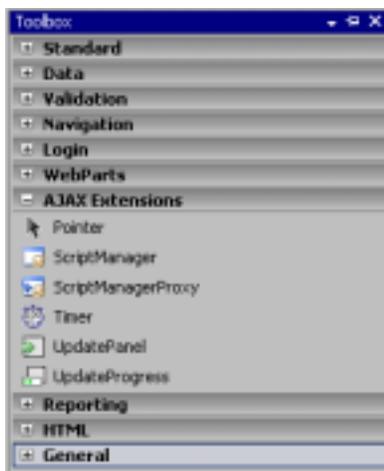


Figure 2-1 The *UpdatePanel* control in the *AJAX Extensions* section

Properties of the UpdatePanel Control

You can view different properties of the **UpdatePanel** control in the **Properties** window. To display it, click on the **UpdatePanel** control and press the F4 key; the **Properties** window will be displayed, as shown in Figure 2-2. Some important properties of the **UpdatePanel** control such as **ChildrenAsTriggers**, **ContentTemplate**, **UpdateMode**, **Triggers**, and **RenderMode** are discussed next.

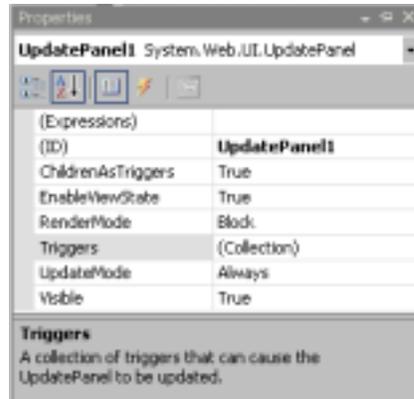


Figure 2-2 The *Properties* window of the *UpdatePanel* control

ChildrenAsTriggers

The **ChildrenAsTriggers** property returns a boolean value, **True** or **False**, which indicates whether the postbacks from a child control will refresh the **UpdatePanel** control or not. By default, this property is set to **True** and you can set this default property to **False** only when the **UpdateMode** property is set to the value **Conditional**. You can set the value of this property either in the **Properties** window or in the code.

For example:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" ChildrenAsTriggers="false">
```

ContentTemplate

The **ContentTemplate** property defines the area that can be updated asynchronously under the **UpdatePanel** control. You can put controls that you want to update asynchronously under the **ContentTemplate** tag. Note that the **ContentTemplate** property is not available in the **Properties** window.

For example:

```
<ContentTemplate>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="Button"></asp:Button>
</ContentTemplate>
```

UpdateMode

You can set the **UpdateMode** property of the **UpdatePanel** control to **Always** or **Conditional**. The default value of the **UpdateMode** property is **Always** that indicates that the **UpdatePanel** control will always update its content during an asynchronous postback. In another case, if you set the value of **UpdateMode** property to **Conditional**, the **UpdatePanel** control will update either its own area or whole page, depending upon the value set in the **ChildrenAsTriggers** property.

For example:

```
<asp:UpdatePanel1 ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
```

Triggers

You can use the **Triggers** property in the **UpdatePanel** control to fire a postback using some specific controls. However, the postback can be either asynchronous or a normal postback. If you want to make the postback as asynchronous, then you need to mention the specific control with **asp:AsyncPostBackTrigger**, which is a property of **Triggers**. The **Triggers** control consists of two properties: **AsyncPostBackTrigger** and **PostBackTrigger**.

AsyncPostBackTrigger

This property is used to define a control that is responsible for updating the **UpdatePanel** control. You can define the name of the control in the **ControlID** member of the **AsyncPostBackTrigger** property. Similarly, you can define the name of the event in the **EventName** member of the **AsyncPostBackTrigger** property.

For example:

```
<Triggers>  
<asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />  
</Triggers>
```

PostBackTrigger

The **PostBackTrigger** property fires postback for the entire web page, instead of a portion of a page. It works like a normal postback and it does not support the **EventName** member.

For example:

```
<asp:PostBackTrigger ControlID="Button1" />
```

RenderMode

The **RenderMode** property is used to set a value that determines whether the content of the **UpdatePanel** control will be enclosed within the `<div>` or `` element of HTML. The **RenderMode** property has two values, **Block** and **Inline**. **Block** is the default value, which ensures that the content of the **UpdatePanel** control is enclosed within a `<div>` element. The **Inline** value ensures that the content of the **UpdatePanel** is enclosed in an `` tag. When a web page is rendered, the **UpdatePanel** control is rendered as either `<div>` or `` element of HTML. Therefore, to determine the rendering method, you need to set

either the **Inline** or **Block** value for the **RenderMode** property. The value that you set for this property is controlled by JavaScript. The example given below shows an **UpdatePanel** control that is rendered as a `` element.

```
<asp:UpdatePanel RenderMode= "Inline">
  <ContentTemplate>
    <!--contents inside the UpdatePanel -->
  </ContentTemplate>
</asp:UpdatePanel>
```

The following example illustrates the use of the **UpdatePanel** control:

Example 1

Create an application using the **UpdatePanel** control to show the asynchronous postback in the browser.

The following steps are required to create this application:

1. Create a new web application with the name **Ch_02**.
2. Open the *Default.aspx* page, rename it to *Exam_01.aspx* and switch to the **Design** view.
3. In the **AJAX Extensions** section of the **Toolbox**, double-click on the **ScriptManager** control; the **ScriptManager** control will be added to the page.
4. Double-click on the **UpdatePanel** control in the **AJAX Extensions** section of the **Toolbox**; the **UpdatePanel** control will be added to the page.
5. Click inside the **UpdatePanel** control and then double-click on the **Label** and **Button** controls in the **Standard** section of the **Toolbox**; the **Label** and **Button** controls will be added to the page inside the **UpdatePanel** control. Figure 2-3 shows the layout of application.
6. Double-click on the **Button** control to switch to the code behind file.
7. Add a line of code in the **Button click** event handler as follows:

Writing Code Using C#

```
protected void Button1_Click(object sender, EventArgs e)      1
{                                                                2
  Label1.Text = DateTime.Now.ToString();                       3
}                                                                4
```

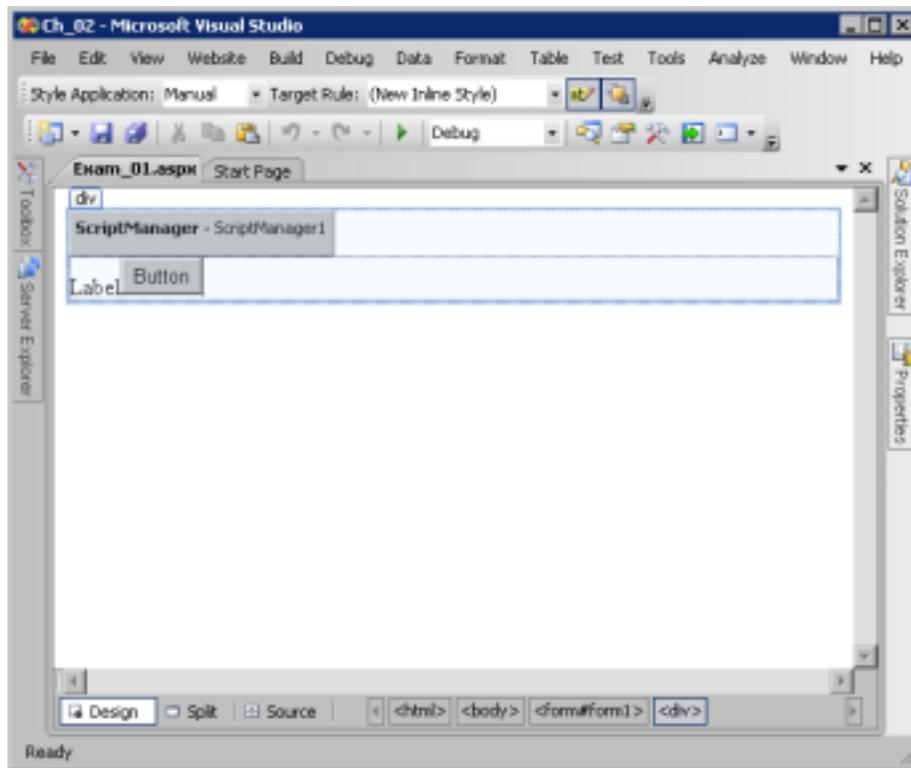


Figure 2-3 The layout of the application

Writing Code Using VB

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Button1.Click

Label1.Text = Date.Now.ToString()

End Sub

8. Save and run the application to see the result on the browser.
9. Click on the **Button** control, the current date and time of the system will be displayed in place of the **Label** control, as shown in Figure 2-4.

Now, each time you click on the **Button** control, only the panel part of the page will be refreshed, and not the whole page. Also, there will be no flickering on the page.

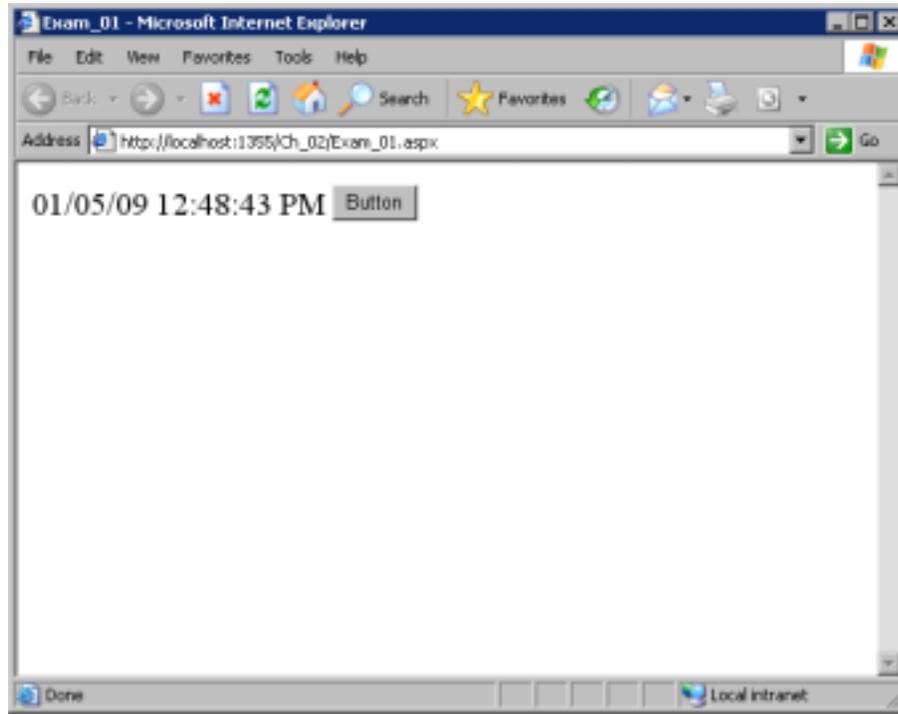


Figure 2-4 The asynchronous update in browser

The source code of the application is as follows:

```

<!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> Exam_01</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
          <ContentTemplate>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
            <asp:Button ID="Button1" runat="server" Text="Button"
            onclick="Button1_Click1" />
          </ContentTemplate>
        </asp:UpdatePanel>
      </div>
    </form>
  </body>
</html>

```

**Note**

*In the previous code, you may have noticed that a new tag, called **ScriptManager**, has been used. You will learn more about this tag in the later chapters.*

Explanation

The line-by-line explanation of the source code is as follows:

Line 6

```
<body>
```

The **<body>** tag defines the document's body.

Line 7

```
<form id= "form1" runat= "server">
```

This line represents the starting of the **<form>** tag for which the code will be written. The first attribute, **id="form1"**, indicates the name of the current form, **form1**. The second attribute, **runat="server"**, indicates that the form will be executed at server-side, which means that the code written under this form will be executed at the server-side.

Line 8

```
<div>
```

This is the default tag. By default, the **<div>** tag will be added in every form.

Line 9

```
<asp:ScriptManager ID= "ScriptManager1" runat= "server"> </asp:ScriptManager>
```

To add the **ScriptManager** tag in the form is compulsory for every AJAX-enabled form. This is the starting and closing tag of the **ScriptManager** control, and is added automatically when you add the **ScriptManager** control from the **Standard** section of **Toolbox** in the **Design** view.

Line 10

```
<asp:UpdatePanel ID= "UpdatePanel1" runat= "server">
```

This line introduces a powerful control of AJAX, called the **UpdatePanel** control. The **asp:UpdatePanel** tag is the starting tag of the **UpdatePanel** control and the **UpdatePanel1** is the name of the **UpdatePanel** control. The **runat="server"** attribute indicates that the **UpdatePanel** control will be executed at the server-side.

Line 11

```
<ContentTemplate>
```

This line indicates the starting of the **<ContentTemplate>** tag. The **<ContentTemplate>** tag works like a container of the **UpdatePanel** control. If you place any control inside the **UpdatePanel** control, it will be added automatically inside the **<ContentTemplate>** tag.

Line 12

```
<asp:Label ID= "Label1" runat= "server" Text= "Label"> </asp:Label>
```

This line indicates that the **Label** control, named **Label1**, has been added inside the **UpdatePanel** control. The **runat="server"** attribute indicates that the **Label** control will be executed at the server-side. The **Text** attribute indicates the caption of the **Label** control that

will be displayed at runtime and the `</asp:Label>` indicates the end tag of the **Label** control.

Line 13

```
<asp:Button ID="Button1"runat="server"Text="Button" onclick="Button1_Click1" />
```

This line indicates that the **Button** control that is added inside the **UpdatePanel** control will be responsible for performing the postback of the form. The **asp:Button** tag indicates the starting tag of the **Button** control and the **ID="Button1"** attribute indicates the name of the **Button** control. The **Text="Button"** attribute indicates the caption of the **Button** control that will be displayed at runtime. And, the **onclick="Button1_Click1"** attribute shows the name of the event that will fire after the **Button1** is clicked at runtime.

Line 14

```
</ContentTemplate>
```

This line indicates the end of the **ContentTemplate** tag.

Line 15

```
</asp:UpdatePanel>
```

This line indicates the end of the **UpdatePanel** tag.

Line 16

```
</div>
```

This line indicates the end of the **div** element.

Line 17

```
</form>
```

This line indicates the end of the **form** tag.

Line 18

```
</body>
```

This line indicates the end tag of the **body** element.

The explanation of code behind file is as follows:

Line 3

```
Label1.Text = DateTime.Now.ToString();
```

This line indicates that the current date and time of the system will be stored in the **Label1** control.

Advantages of Using the UpdatePanel Control

The **UpdatePanel** control enables you to build client-centric web applications that are not only up-to-date but also fulfill the client's requirements in moments. The main advantage of using the **UpdatePanel** control is that it updates a particular area of the page rather than the whole page. When the **UpdatePanel** control updates a particular area of the page, it is known as the partial page update. This control reduces the amount of flickering of the page that normally occurs when a web page performs postbacks.

Performing Updates with Triggers

The default behavior of the **UpdatePanel** control is to update the child controls that normally trigger a postback. This is because the default value of the **UpdateMode** property is **Always**, whereas the default value of the **ChildrenAsTriggers** property is **True**. The **UpdateMode** property is a type of **UpdatePanelUpdateMode** enumeration. As you learned earlier, the possible values of the **UpdateMode** property are, **Always** and **Conditional**. The default value is **Always**, which means that the **UpdatePanel** control will be refreshed each time a postback occurs.

Table 2-1 summarizes the results of possible combinations of the **UpdateMode** and **ChildrenAsTriggers** properties.

UpdateMode	ChildrenAsTriggers
Always	False
Always	True
Conditional	False
Conditional	True

*Table 2-1 Different values of the **UpdateMode** property of the **UpdatePanel** control*

The first combination of Table 2-1 will result in error. This is because, on the one hand, you are prompting the **UpdatePanel** control to always update its content, and on the other hand, you are prompting the triggering of child control to be **False**. So, the result will be an error. In the second combination, the value of the **UpdateMode** property is **Always** and the value of the **ChildrenAsTriggers** property is **True**, which means the **UpdatePanel** control will refresh or update its content in each case. In the third combination, the value of the **UpdateMode** property is **Conditional** and the value of the **ChildrenAsTriggers** property is **False**. It means that the **UpdatePanel** control will update its content, with a condition that a control from outside the panel triggers a postback. In the last combination, the value of the **UpdateMode** property is **Conditional** and the value of the **ChildrenAsTriggers** property is **True**. It means that the **UpdatePanel** control will update its content either when the child control triggers a postback or a control from outside the panel triggers a postback.

The following example illustrates the triggering of the **UpdatePanel** control:

Example 2

Create an application to show the asynchronous postback in the browser using the **Trigger** control.

The following steps are required to create this application:

1. Open the **Ch_02** application. Right-click on the icon of the **Ch_02** application in the **Solution Explorer** window and choose the **Add New Item** option from context menu; the **Add New Item** dialog box will be displayed.
2. In the **Add New Item** dialog box, select **Web Form** from **Templates** area. Next, change the name of form to *Exam_02.aspx* and then choose the **Add** button; the **form** will be added to your application.
3. Double-click on the **ScriptManager** and **UpdatePanel** controls in the **AJAX Extensions** section to add these controls to the form.
4. Click inside the **UpdatePanel** control and then double-click on the **Label** control in the **Standard** section of the **Toolbox**. Next, change the text of the **Label1** control to **Inside Label**.
5. Click outside the **UpdatePanel** control and then double-click on one more **Label** control and a **Button** control in the **Standard** section of the **Toolbox**. Next, change the value of the **Text** property of the **Label2** control to **Outside Label**. Figure 2-5 shows the layout of the application.
6. Double-click on the **Button** control to switch to the event handler of the button and then add the following code to the **Button** control event handler.

Writing Code Using C#

```
protected void Button1_Click(object sender, EventArgs e)      1
{                                                            2
    Label1.Text = DateTime.Now.ToString();                    3
    Label2.Text = DateTime.Now.ToString();                    4
}                                                            5
```

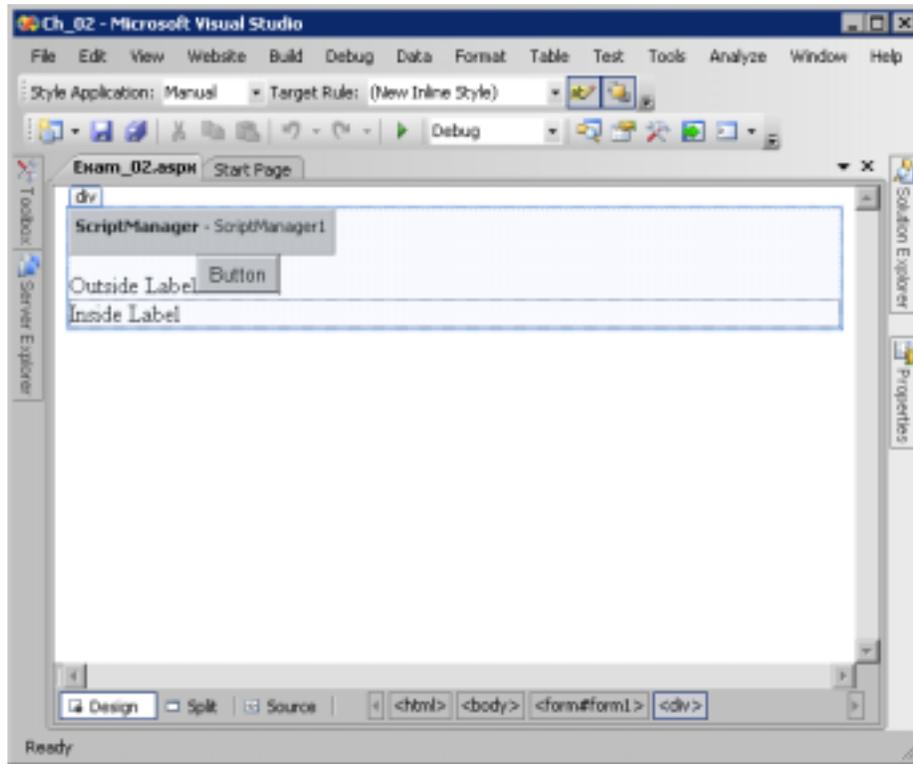


Figure 2-5 The layout for performing updates using Triggers

Writing Code Using VB

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Button1.Click

Label1.Text = Date.Now.ToString()
Label2.Text = Date.Now.ToString()

End Sub

- Return to the **Design** view and then click on **Source**. Next, add lines from 9 to 11 inside the **<Triggers>** tag. The code will be explained in the next section.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">           1
<html xmlns="http://www.w3.org/1999/xhtml">                             2
<head runat="server">                                                 3
<title> Exam_02</title>                                               4
</head>                                                                5
<body>                                                                6
<form id="form1" runat="server">                                       7

```

```

<div> 8
<asp:ScriptManager ID="ScriptManager1" runat="server"> 9
</asp:ScriptManager> 10
<asp:Label ID="Label2" runat="server" Text="Outside Label"></asp:Label> 11
<asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Button" /> 12
<asp:UpdatePanel ID="UpdatePanel1" runat="server"> 13
<ContentTemplate> 14
<asp:Label ID="Label1" runat="server" Text="Inside Label"></asp:Label> 15
</ContentTemplate> 16
<Triggers> 17
<asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" /> 18
</Triggers> 19
</asp:UpdatePanel> 20
</div> 21
</form> 22
</body> 23
</html> 24

```

8. Save your application and execute it by pressing the F5 key; the output will be displayed in the browser.
9. Click on the **Button** control. You will note that only **InsideLabel** will show the current date and time of system, but **OutsideLabel** will not be refreshed. The output of the application is shown in Figure 2-6.

Explanation

The line-by-line explanation of the source code given earlier is as follows:

Line 17

<Triggers>

This line indicates the starting tag of the **Trigger** control.

Line 18

<asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />

This line indicates the properties of the **<Triggers>** tag. The property **asp:AsyncPostBackTrigger** indicates that the **Button1** control will be responsible for updating the **UpdatePanel** control. The **ControlID** attribute is a member of **AsyncPostBackTrigger** property and it specifies the name of the control that will be responsible for postback. Similarly, the **EventName** member indicates the name of the event of the control.

Line 19

</Triggers>

This line indicates the closing tag of the **Trigger** control.

The explanation of code behind file is as follows:

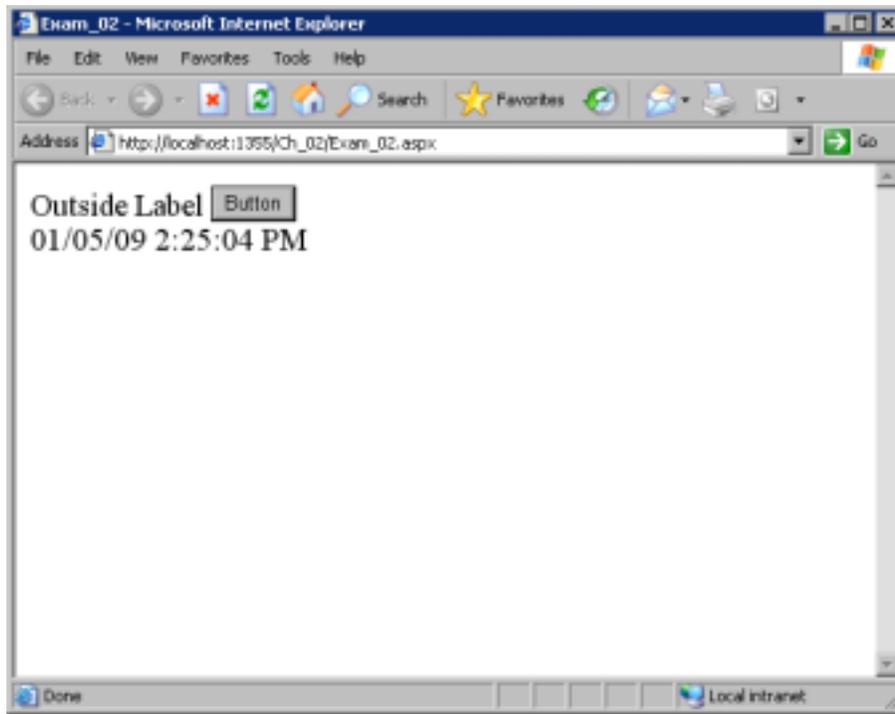


Figure 2-6 Performing updates using Triggers

Line 3

Label1.Text = DateTime.Now.ToString();

This line indicates that the current date and time of system will be stored in the **Label1** control.

Line 4

Label2.Text = DateTime.Now.ToString();

This line indicates that the current date and time of system will be stored in the **Label2** control.

In Example 1, you learned that the **UpdatePanel** control was refreshed by calling the postback of a button control that was inside the **UpdatePanel** control. Note that in this example, the **UpdatePanel** control refreshes its content by calling the postback of a button control that is outside the **UpdatePanel** control. It occurs with the help of the **Trigger** control. In lines 6 and 7 of the source code given above, you will notice that the **Label** and **Button** controls are written outside the **UpdatePanel** control. But again, these controls perform only the asynchronous postback because of the **Trigger** control. The **Trigger** control enables you to perform asynchronous postback on a button or specific controls. Also, you do not have to put postback controls like **Button** inside the **UpdatePanel** control for updating the panel. You need to only specify the name of the control for the **ControlID** attribute as well as the name of the event for the attribute **EventName**.

Using Multiple UpdatePanel Controls

You can systematically update portions of a page separately or together. It does not matter how many **UpdatePanel** controls you use in a single page and therefore, you can perform multiple updates on a single page.

The following example illustrates the use of multiple **UpdatePanel** controls:

Example 3

Create an application to show the asynchronous updates of multiple portions in the browser.

The following steps are required to create this application:

1. Open the **Ch_02** application. Add a new web form to the application from the **Solution Explorer** window.
2. Change the name of the form to *Exam_03.aspx*.
3. In the **AJAX Extensions** section, double-click on the **ScriptManager** control to add it to the page.
4. Add two **UpdatePanel** controls to the page.
5. Click inside the first **UpdatePanel** control and add a **Label** control.
6. Similarly, click inside the second **UpdatePanel** control and add a **Label** control to it.
7. Now, click inside any of the **UpdatePanel** controls and add a **Button** control from the **Standard** section of the **Toolbox**.



Note

You can add a **Button** control inside any of the **UpdatePanel** controls. After performing the postback using the **Button** control, both the **UpdatePanel** controls will be updated.

8. Change the value of the **Text** property of the **Button** control to **Update Again!** in the **Properties** window.
9. Click outside the area of the **UpdatePanel** control and then add one more **Label** control from the **Standard** section of the **Toolbox** to the page. The layout of the page after adding the controls will be as shown in Figure 2-7.

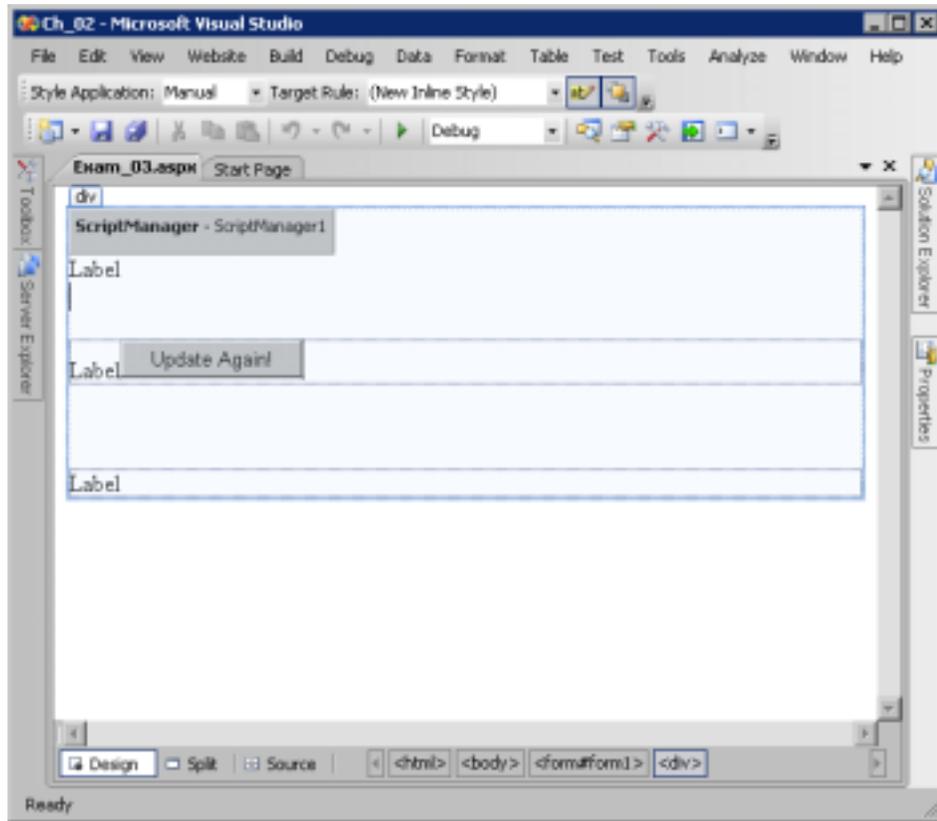


Figure 2-7 The layout of multiple *UpdatePanel* controls

10. Double-click on the page to switch into the **Page_Load** event. Add the following code to the **Page_Load** event:

Writing Code Using C#

```
protected void Page_Load(object sender, EventArgs e)           1
{                                                                 2
    Label1.Text = "First Panel Updated at:" + DateTime.Now.ToLongTimeString(); 3
    Label2.Text = "Second Panel Updated at:" + DateTime.Now.ToLongTimeString(); 4
    Label3.Text = "Outside Panel Updated at:" + DateTime.Now.ToLongTimeString(); 5
}                                                                 6
```

Writing Code Using VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
Handles Me.Load
```

```
Label1.Text = "First Panel Updated at:" + DateTime.Now.ToLongTimeString()  
Label2.Text = "Second Panel Updated at:" + DateTime.Now.ToLongTimeString()  
Label3.Text = "Outside Panel Updated at:" + DateTime.Now.ToLongTimeString()
```

```
End Sub
```

11. Save the application and execute it. You will notice that all three **Label** controls have been updated and display the updated time. Figure 2-8 shows the output of the application.
12. Now, click on the **Update Again!** button. On doing so, you will notice that the **Label** control outside the **UpdatePanel** controls is not updated, while the **Label** controls inside the **UpdatePanel** controls are updated.

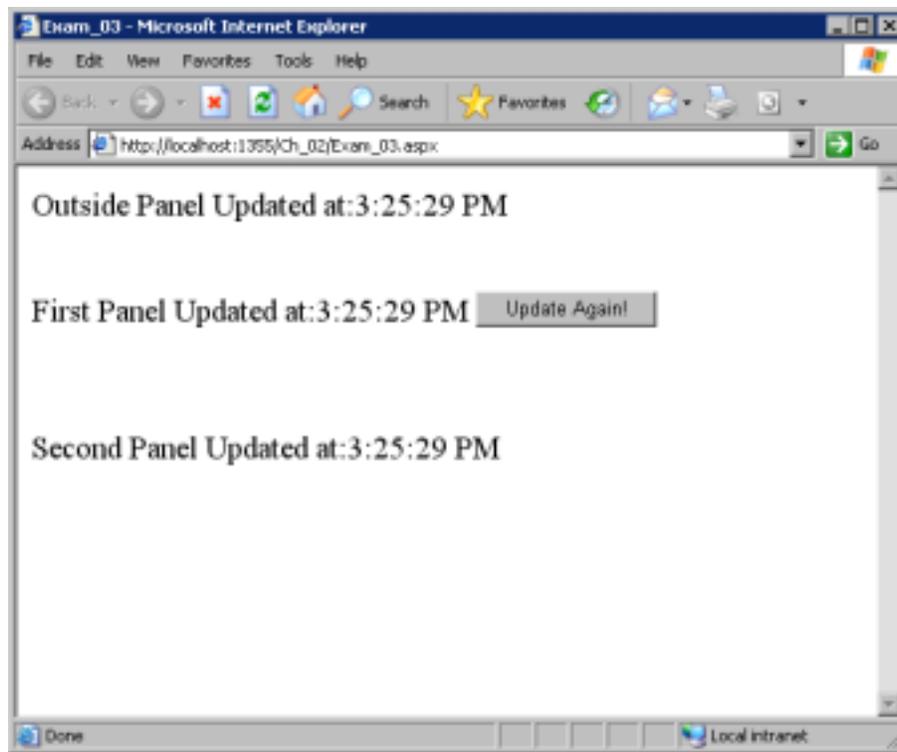


Figure 2-8 Multiple **UpdatePanel** controls updated

Note that both the Labels inside the **UpdatePanel** controls show the updated time. Which proves that they have been updated simultaneously. It means that multiple portions of a page can be updated simultaneously. In this example, you updated only two portions of a

page simultaneously. However, you can also use this method to update a number of portions on a single page.

Updating the Master Page Using the UpdatePanel Control

You can update the content pages of a **master page** with the help of the **UpdatePanel** control. In such cases, you do not have to apply any new concept to implement updates on the **master page**. The technical advantage while applying updates on a **master page** using this method is that you do not need to add the **ScriptManager** control for the content pages. You can simply add the **ScriptManager** control to the **master page** and then you will get its effect on all the content pages added to it. The **master page** provides the **ScriptManager** control for all content pages, so you can perform partial page updates for these pages as well.

The following example illustrates a **master page** being updated using the **UpdatePanel** control:

Example 4

Create an application to update the **master page** using the **UpdatePanel** control.

The following steps are required to create this application:

1. Open the **Ch_02** application. Right-click on the icon of the **Ch_02** application in the **Solution Explorer** window and choose the **Add New Item** option from context menu; the **Add New Item** dialog box will be displayed.
2. In the **Add New Item** dialog box, choose the **Master Page** option from the **Templates** area. Next, choose the **Add** button; the **master page** with the name *MasterPage.master* will be added to your project.
3. Double-click on the **ScriptManager** control in the **AJAX Extensions** section to add it to the **master page**. Make sure the **ScriptManager** control is added to your page outside the **ContentPlaceHolder** control, and it should be placed at the top of all controls, as shown in Figure 2-9.



Note

The *ContentPlaceHolder* control is automatically added to the *master page*. It defines a region to place contents in the ASP.NET *master page*.

4. Now, right-click on the **MasterPage.master** node in the **Solution Explorer** window, as shown in Figure 2-10, and then select the **Add Content Page** option from the context menu; a default *aspx* page will be added to your application. This *aspx* page is called the **Content** page.
5. Change the name of the page to *First.aspx*. Next, click inside the **ContentPlaceHolder** control of the *First.aspx* page and double-click on the **UpdatePanel** control in the **Standard** section of the **Toolbox** to add it to the content area of the *First.aspx* page.

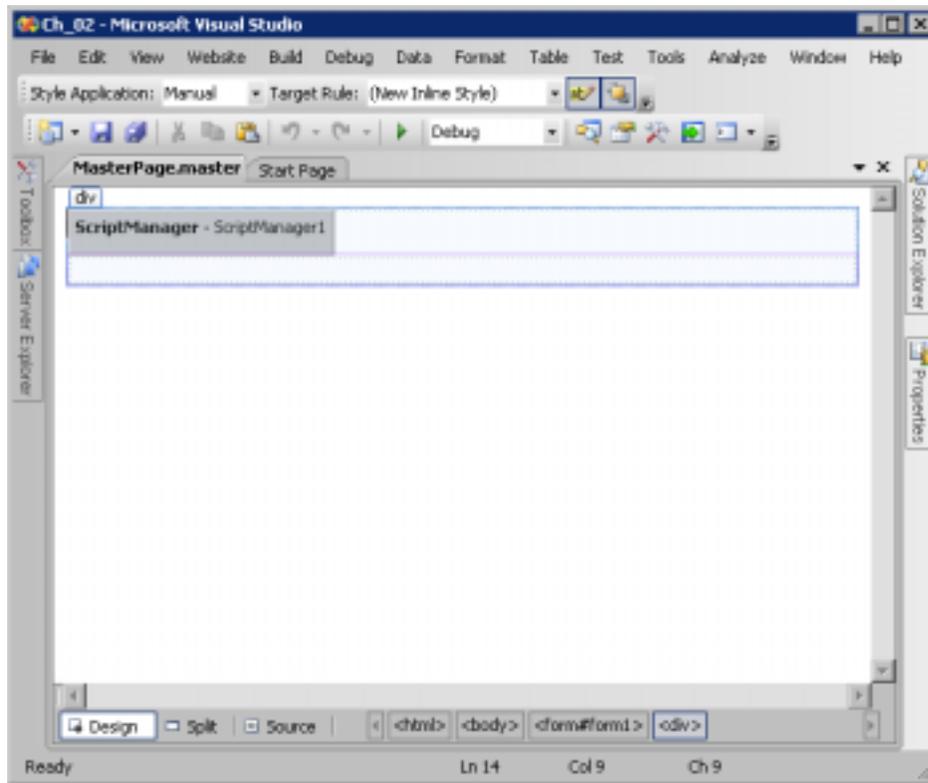


Figure 2-9 The layout of a master page

6. Click inside the **UpdatePanel** control and then double-click on the **Calendar** control in the **Standard** section of the **Toolbox**. Figure 2-11 shows the layout of the **master page** with the **Calendar** control added to it.
7. Repeat the Step 5, and change the name of the page to *Second.aspx*. Next, click inside the **ContentPlaceHolder** control of the *Second.aspx* page and double-click on the **UpdatePanel** control in the **Standard** section of the **Toolbox** to add it to the content area of the *Second.aspx* page.
8. In the *Second.aspx* page, click inside the **UpdatePanel** control then, double-click on the **Label** and **Button** controls in the **Standard** section of the **Toolbox**. Figure 2-12 shows the **master page** with the **Label** and **Button** controls added to it.
9. Double-click on the *Second.aspx* page to switch to the **Page_Load** event handler of the form and add the following source code:

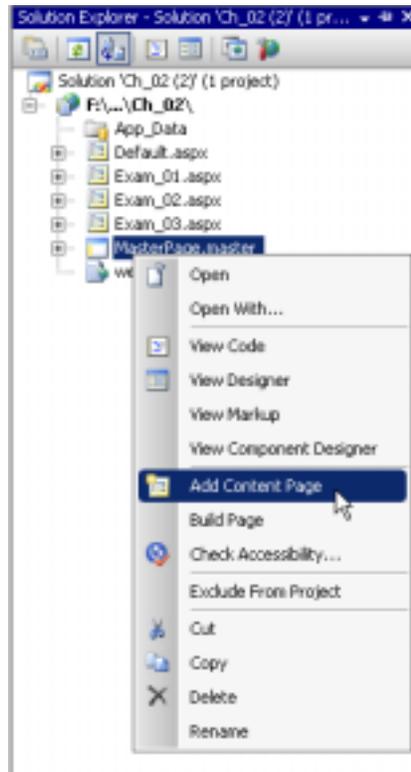


Figure 2-10 Adding Content Page in master page

Writing Code Using C#

```
protected void Page_Load(object sender, EventArgs e)           1
{                                                                 2
    Label1.Text = DateTime.Now.ToString();                       3
}                                                                 4
```

Writing Code Using VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Me.Load

    Label1.Text = Date.Now.ToString()

End Sub
```

10. Now, open the *First.aspx* page and execute it; the browser will be displayed, as shown in Figure 2-13.
11. Click on the forward or backward button of the calendar to change month. You will notice that there is no flickering when the page refreshes.

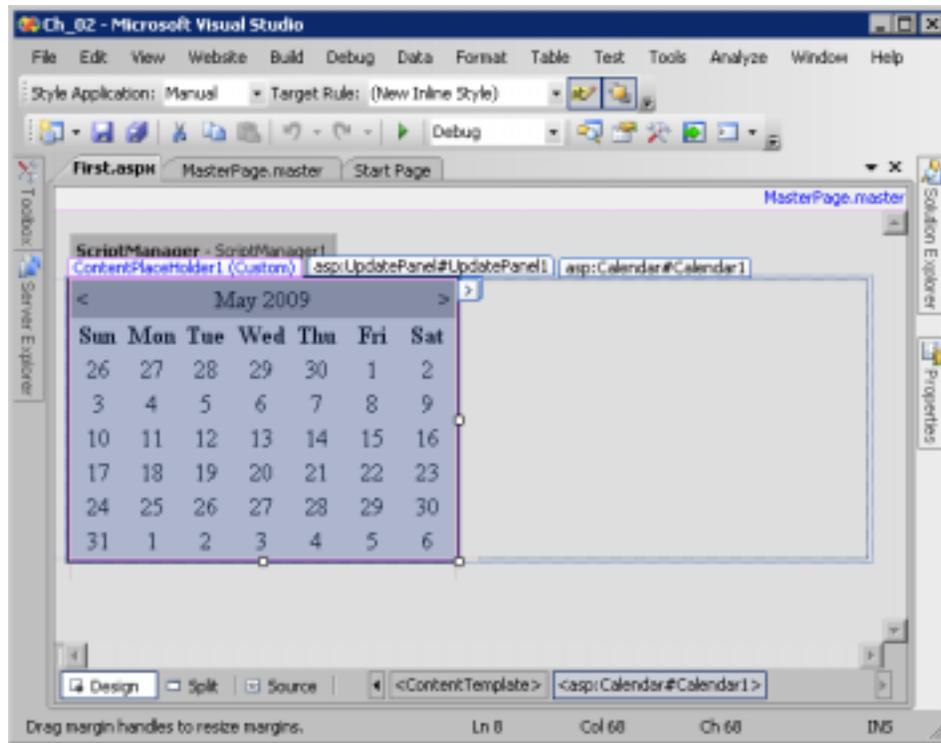


Figure 2-11 The master page layout with the Calendar control

12. Close the browser. Now, open the *Second.aspx* page and execute it. You will see the browser, as shown in Figure 2-14.
13. Click on the **Button** control. You will notice that in this case also there is no flickering and the page is refreshed.

In this example, you learned how partial page updates occur on a **master page**. In such cases, you cannot only update portions of a content page but also multiple content pages of the **master page**.

Limitations of the UpdatePanel Control

The **UpdatePanel** control has certain limitations that are described as follows:

1. In case of validation controls, AJAX registers those client scripts, which are not compatible with the **UpdatePanel** control. So, to use the validation controls with JavaScript, you will have to disable the client script.

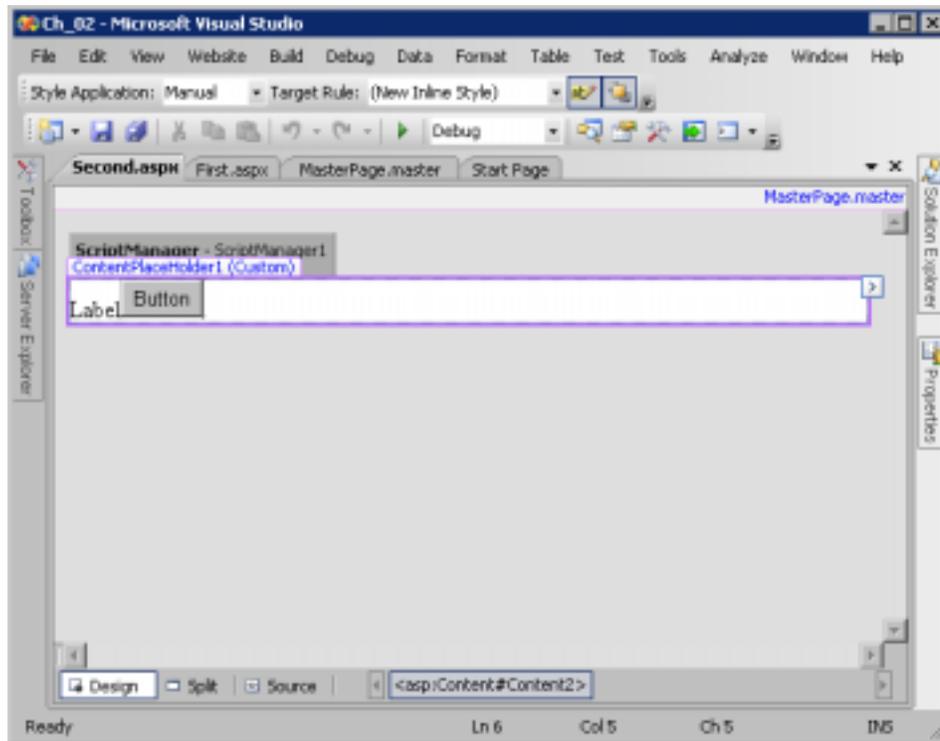


Figure 2-12 The master page layout with the *Label* and *Button* controls

You can disable the validator controls by using the code given below:

```
<asp:RequiredFieldValidator EnableClientScript="false" runat="server"
ControlToValidate="TextBox1" />
```

When you write the above code, the validator controls will not validate the client-side script.

- Another control that does not work with the **UpdatePanel** control is the **FileUpload** control. This control will not work when you use it inside the **UpdatePanel** control and call a postback asynchronously. Therefore, in such a case, you need to use the **PostBackTrigger** property instead of the **asyncPostBackTrigger** property under the **Triggers** tag to make it work with the **UpdatePanel** control.

The following code shows the use of the **PostBackTrigger** property of the **UpdatePanel** control:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="conditional"> 1
<contentTemplate> 2
<asp:FileUpload runat="server" ID="fileUpload"/> 3
<asp:Button runat="server" Text="Upload" ID="UploadButton" /> 4
</ContentTemplate> 5
```

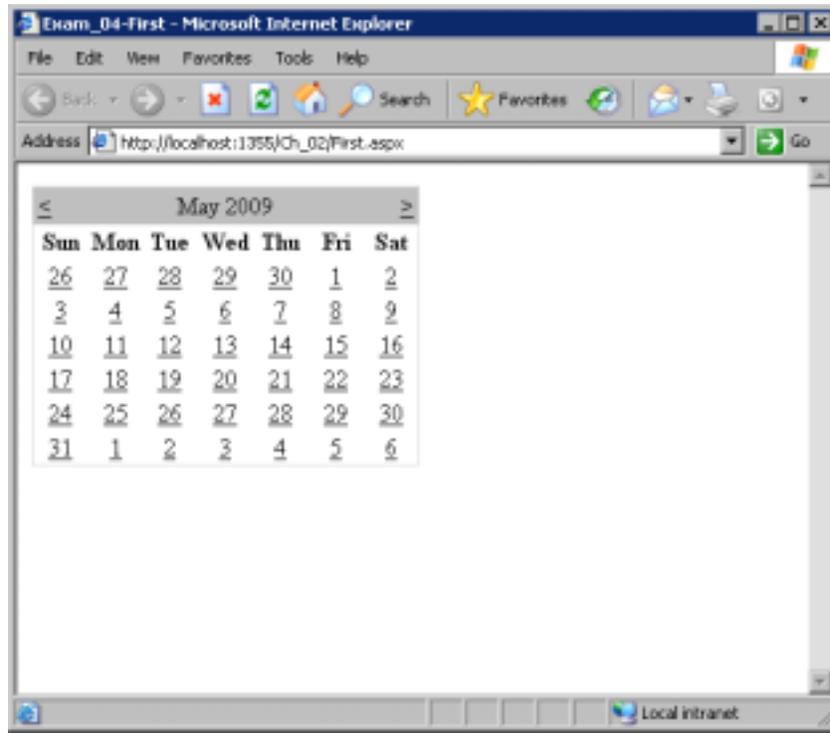


Figure 2-13 Updating the Calendar control on the master page

```

< Triggers>
< asp:PostBackTrigger ControlID= "UploadButton" />
</Triggers>
</asp:UpdatePanel>

```

6
7
8
9



Note

The *PostBackTrigger* property does not support the *EventName* attribute.

Explanation

The line-by-line explanation of the above source code is as follows:

Line 3

```
<asp:FileUpload runat="server" ID="fileUpload"/>
```

This line indicates that the **FileUpload** control will be added inside the **UpdatePanel** control. The **runat="server"** attribute will execute the control at server-side. Moreover, the **ID="fileUpload"** attribute indicates that **fileUpload** is the name of the **FileUpload** control.

Line 4

```
<asp:Button runat="server" Text="Upload" ID="UploadButton" />
```

This line indicates that a **Button** control has been added to the page, which will be responsible for performing postback.

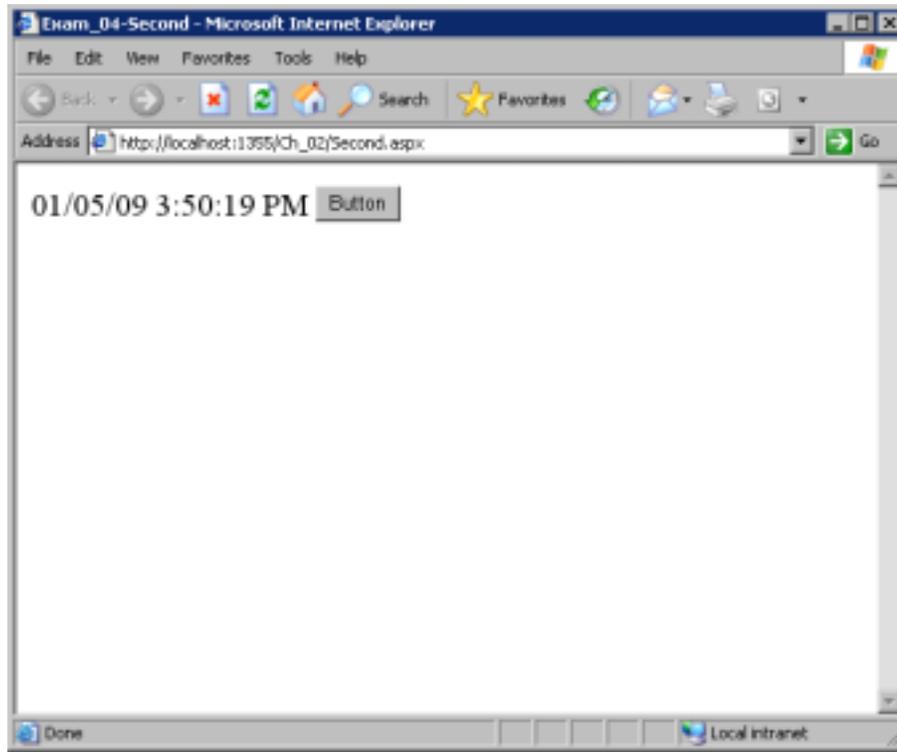


Figure 2-14 Updating the *Label* control on the master page

Line 7

```
<asp:PostBackTrigger ControlID="Button1" />
```

This line indicates the properties of the **Triggers** control. **asp:PostBackTrigger** indicates that a postback will be performed by the **Button** control, and **ControlID="Button1"** indicates that the name of the **Button** control will be **Button1**.

THE UpdateProgress CONTROL

When you use the **UpdatePanel** control, an asynchronous postback is performed and the **UpdatePanel** control is refreshed without causing any flickering or disturbances for the user. In this case, although the postback is carried out in the usual way by sending request to server and receiving response from it, but this postback is performed so fast that it becomes impossible for the user to notice its progress. Even though it provides a good service, but is not user-friendly. Whenever the user updates the portion of a page to refresh the **UpdatePanel** control, he cannot see the progress of the updates. To solve this problem, you can use the **UpdateProgress** control that creates a graphical user interface to show the progress.

The **UpdateProgress** control shows progress whenever it is refreshed. Basically, this control allows you to display UI on the page while the postback is going on. Also, it allows a user to cancel callback, if a user does not want postback for the **UpdatePanel** control.

The following code snippet is used to declare the **UpdateProgress** control:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"> 1
<ProgressTemplate> 2
<div class="progress"> 3
 4
</ProgressTemplate> 5
</asp:UpdateProgress> 6
```

Explanation

The line-by-line explanation of the source code given above is as follows:

Line 1

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
```

This line indicates the starting tag of the **UpdateProgress** control. Here, the **ID="UpdateProgress1"** attribute indicates that **UpdateProgress1** will be the unique name of the **UpdateProgress** control in the application. Similarly, the **runat="server"** attributes indicates that the **UpdateProgress** control will execute at server-side.

Line 2

```
<ProgressTemplate>
```

This line indicates the properties of the **UpdateProgress** control. The **ProgressTemplate** property will be explained later in this chapter.

Line 3

```
<div class="progress">
```

This line indicates that a **<div>** element is being added inside the **UpdateProgress** control that will adjust the image of the progress that you want to display.

Line 4

```

```

This line indicates that an image will be added inside the **UpdateProgress** control that will show the status of the progress. You can specify the full path of the image under the **src** attribute.

Line 5

```
</ProgressTemplate>
```

This line indicates the end of the **<ProgressTemplate>** tag.

Line 6

```
</asp:UpdateProgress>
```

This line indicates the end of the **UpdateProgress** control.

Properties of the UpdateProgress Control

Some important properties of the **UpdateProgress** control are **ProgressTemplate**, **AssociatedUpdatePanelID**, **DisplayAfter**, and **DynamicLayout**. These properties are discussed next.

ProgressTemplate

The **ProgressTemplate** property defines the template where you can create an interface to show the progress of updating the **UpdatePanel** control. When a callback occurs, the content inside the **ProgressTemplate** property will be displayed on the browser. If you do not include anything between the **<ProgressTemplate>** and **</ProgressTemplate>** tags, the **UpdateProgress** control will not display anything.

AssociatedUpdatePanelID

It refers to the **UpdatePanel** control that you want to associate with the **UpdateProgress** control. When a postback event is generated inside the **UpdatePanel** control, the associated **UpdateProgress** control will be displayed. If you do not specify the **AssociatedUpdatePanelID** property, the **UpdateProgress** control will be displayed for any asynchronous postback control inside the **UpdatePanel** or the **Triggers** control.

DisplayAfter

You can set value of time delay in progress of **UpdatePanel** control in millisecond to display the contents of **ProgressTemplate**.

DynamicLayout

It determines whether the **ProgressTemplate** property is rendered dynamically.

The following example illustrates the use of the **UpdateProgress** control:

Example 5

Create an application to show the progress of updating the record asynchronously in the browser.

The steps required to use the **UpdateProgress** control are given next. Note that before creating the application, in this case, you need to create a table in SQL server:

Steps for Creating the Table in SQL Server

1. Open **SQL Server Management Studio** from the **Programs** menu. Next, connect to **SQL** by specifying the **Server name**, **Login**, **Password**, and so on.
2. Right-click on the **Databases** folder and then choose the **New Database...** item in the shortcut menu; the **New Database** dialog box will be displayed.
3. Enter the name of the database, for example, **AJAX**, in the **Database name:** field. Then, choose the **OK** button; a database with the name **AJAX** will be created under the **Databases** folder.

4. Now, double-click on the **AJAX** database. You will see some folders under the **AJAX** database. Right-click on the **Tables** folder and then choose the **New Table...** option from the shortcut menu; a table will be displayed.
5. Enter the name of fields and data types under the **Column Name** and **Data Type** fields, respectively, refer to Table 2-2.

Name of Fields	Data Type	Allow Nulls
empID	smallint	No
empName	varchar(20)	Yes
empSalary	decimal(18,0)	Yes

Table 2-2 The employee table in the design mode

6. Save the table by clicking on the **save** icon in the Toolbar; you will be prompted to enter the name of the table. Enter **employee** and then choose the **OK** button; the **employee** table will be created.
7. Now, double-click on the **Tables** folder and then right-click on the **dbo.employee** table. Next, click on the **Open Table**; the **employee** table will open.
8. Now, enter some records in each field of the **employee** table, refer to Table 2-3.

empID	empName	empSalary
1	John	4000
2	Steven	3000
3	Clark	5000
4	Suzenne	4000
5	Thomas	5000

Table 2-3 The employee table with records

9. Save the table by clicking on the **save** icon in the toolbar. Figure 2-15 shows the final table created. Now, close the **Management Studio**.

Table - dbo.employee			
	empID	empName	empSalary
▶	1	John	4000
	2	Steven	3000
	3	Clark	5000
	4	Suzanne	4000
	5	Thomas	5000
*	NULL	NULL	NULL

Figure 2-15 The employee table created in SQL

Steps to Create the Application

1. Open the **Ch_02** application. Add a new web form to the application from the **Solution Explorer** window.
2. Change the name of the form to *Exam_05.aspx*.
3. Double-click on the **ScriptManager** control in the **AJAX Extensions** section to add it to the form.
4. Double-click on the **UpdatePanel** control in the **AJAX Extensions** section to add it to the form.
5. Click inside the **UpdatePanel** control, and then double-click on the **GridView** control in the **Data** section to add it inside the **UpdatePanel** control.
6. Click anywhere outside the **UpdatePanel**, control and then double-click on the **TextBox** and **Button** controls in the **Standard** section of the **Toolbox** to add them to the form for searching database.
7. Now, double-click on the **UpdateProgress** control in the **AJAX Extensions** section to add it to the form.
8. Next, double-click on the **SqlDataSource** control in the **Data** section to add it to the form. Figure 2-16 shows the layout of the form.
9. Double-click on the *web.config* file in the **Solution Explorer** window to open it.
10. Add the following code after the `</appSettings>` tag in the *web.config* file.

```

<connectionStrings>                                     1
<add name="Connect"connectionString="Server=.;database= AJAX;uid=sa;
pwd=""/>                                                 2
</connectionStrings>                                    3

```

11. Now, switch back to the *Exam_05.aspx* page, click on **Source**, and add the following code:

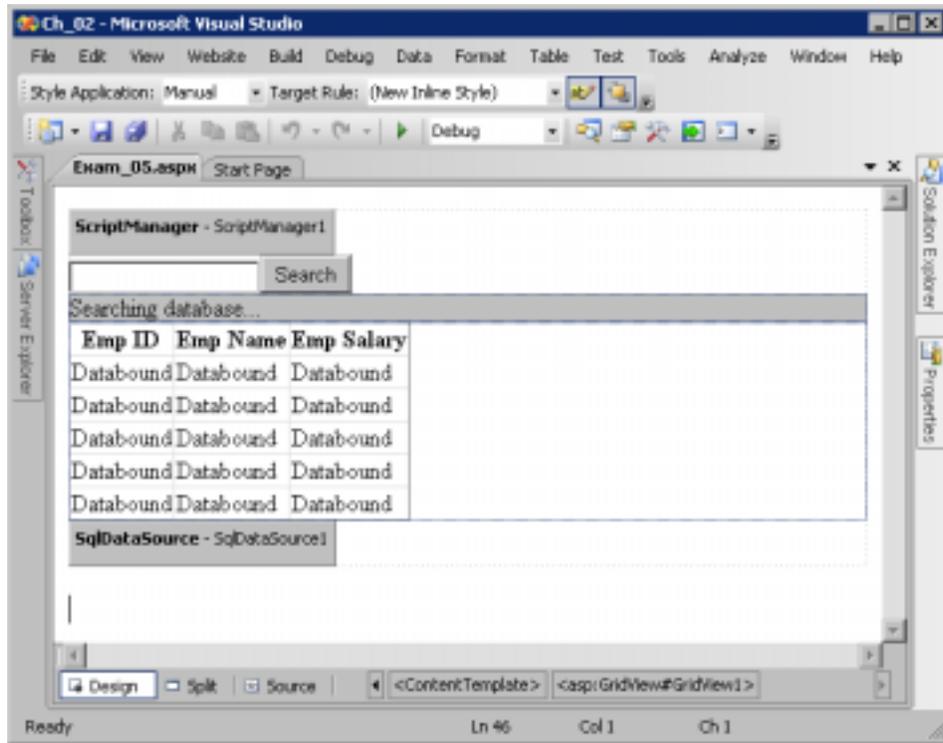


Figure 2-16 The layout for searching database

```

<!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> Exam_05</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:TextBox ID="TextBox1" AutoPostBack="true" runat="server">
</asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Search" />
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
<ProgressTemplate>
<div style="background-color:Fuchsia"> Searching database...</div>
</ProgressTemplate>
</asp:UpdateProgress>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

```

<asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="false" DataSourceID="SqlDataSource1"
EmptyDataText="No records found">
  <Columns>
    <asp:BoundField DataField="empId" HeaderText="Emp ID"
ReadOnly="true" SortExpression="empId" />
    <asp:BoundField DataField="empName" HeaderText="Emp Name"
SortExpression="empName" />
    <asp:BoundField DataField="empSalary" HeaderText="Emp Salary"
SortExpression="empSalary" />
  </Columns>
</asp:GridView>
</ContentTemplate>
</asp:UpdatePanel>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString=
"<%$ ConnectionStrings:Connect %>" ProviderName=
"<%$ ConnectionStrings:Connect.ProviderName %>" SelectCommand=
"select empId, empName, empSalary from employee where (empName like
" + @empName + '%" ">
  <SelectParameters>
    <asp:ControlParameter ControlID="TextBox1" Name="empName"
PropertyName="Text" Type="string" />
  </SelectParameters>
</asp:SqlDataSource>
</div>
</form>
</body>
</html>

```

12. Press the F5 key to run the application; the output will be displayed in the browser.
13. Type the first letter of an employee's name and then click on the **Search** button. Your browser will appear, as shown in Figure 2-17.

Figure 2-17 shows records of employees, whose names start with the letter **s**. During this process, you will notice that when you click on the **Search** button, a message is displayed on your screen, as shown in Figure 2-18. The duration of display of the message on the browser will depend on the system configuration. This message is a part of the **UpdateProgress** control. It is included in the HTML **<div>** element inside the **ProgressTemplate** tag of the **UpdateProgress** control, refer to line 16. It shows the progress of updating the **UpdatePanel** control with the help of the **UpdateProgress** control. Also, the message specified in the **ProgressTemplate** tag will be displayed, as per the design specified by you while coding.

In Figure 2-17, you can notice that records are displayed according to the letter typed in text box. This is a simple application that finds records from database based on certain criteria, and then displays them on the browser. Note that, the aim of creating this application is not to show the records but to display how the **UpdateProgress** control can be used for showing the progress of an application.

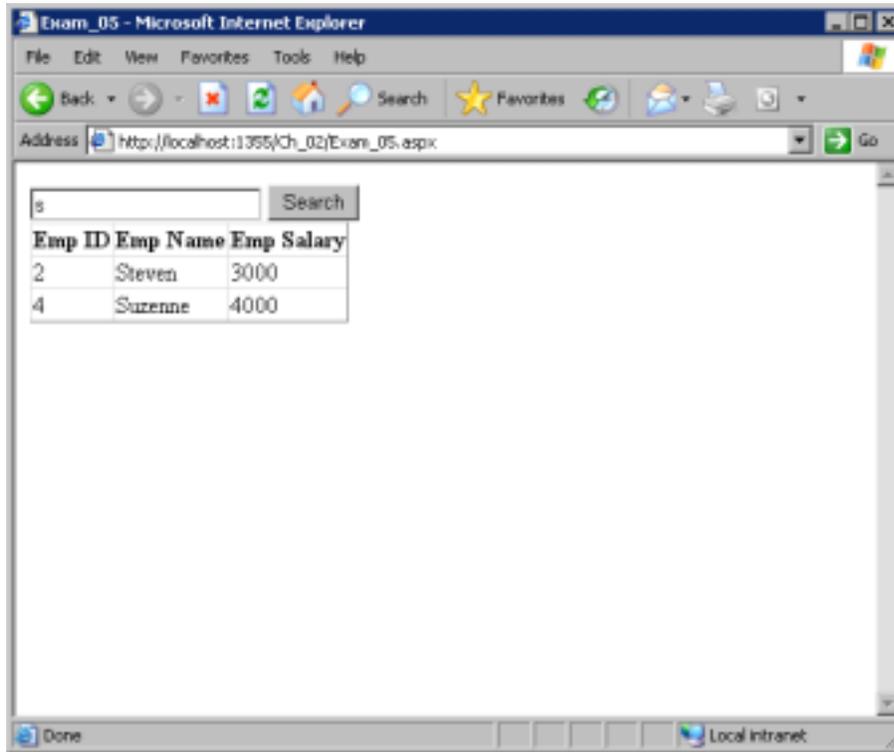


Figure 2-17 Searching database through the *UpdateProgress* control

Searching database...

Figure 2-18 Searching database

Explanation

The line-by-line explanation of the *web.config* code is as follows:

Line 1

<connectionStrings>

This line indicates the start tag of the **<connectionStrings>** element. This element is used to create connection to a database and can be used globally.

Line 2

<add name="Connect" connectionString="Server=.;database=AJAX;uid=sa;pwd="/>

This line indicates the **<add>** element, which is used inside the **<connectionStrings>** element. In **name="Connect"**, the term **Connect** is the name, which will be used as a reference for the **connectionStrings** object. **connectionString** is the connection parameter that will pass to the **ConnectionString** object. It may vary from system to system depending upon the server and database configuration.

Line 3

```
</connectionStrings>
```

This is the end tag of the `<connectionStrings>` element.

The line-by-line explanation of the source code given in Step 11 is as follows:

Line 14

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
```

This line indicates the starting tag of the `UpdateProgress` control. Here, the `ID="UpdateProgress1"` attribute indicates that `UpdateProgress1` is the name of the `UpdateProgress` control. And, the `runat="server"` attribute indicates that the `UpdateProgress` control will execute at server-side.

Line 15

```
<ProgressTemplate>
```

This line indicates the starting tag of the `<ProgressTemplate>` element. This element is used to define a template to show the progress of updating of the `UpdatePanel` control.

Line 16

```
<div style="background-color:Fuchsia">Searching database...</div>
```

In this line, the message `Searching database...` is designed under the `<div>` element, which will be displayed at the time of updating the `UpdatePanel` control.

Line 17

```
</ProgressTemplate>
```

This line indicates the end tag of the `<ProgressTemplate>` element.

Line 18

```
</asp:UpdateProgress>
```

This line indicates the end tag of the `UpdateProgress` control.

Line 19

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

This line adds the `UpdatePanel` control to the form. Here, the `ID="UpdatePanel1"` attribute indicates that the name of the `UpdatePanel` control will be `UpdatePanel1`. And, the `runat="server"` attribute indicates that the `UpdatePanel` control will execute at server-side.

Line 20

```
<ContentTemplate>
```

This line indicates the contents that you want to put inside the `UpdatePanel` control. When the postback fires for the `UpdatePanel` control, the contents inside the `<ContentTemplate>` will be refreshed.

Line 21

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false" DataSourceID="SqlDataSource1" EmptyDataText="No records found">
```

This line adds the `GridView` control inside the `UpdatePanel` control. The

ID=“GridView1” attribute indicates that **GridView1** will be the name of the **GridView** control. Similarly, the **runat=“server”** attribute indicates that the **GridView** control will execute at server-side. The **AutoGenerateColumns=“false”** attribute indicates that extra columns will not be generated automatically at runtime. The **DataSourceID=“SqlDataSource1”** attribute indicates that **SqlDataSource1** will be the name of **DataSource** for **GridView** control. The **EmptyDataText=“No records found”** attribute indicates that the message **No records found** will be displayed when there is no record in the database.

Line 22

<Columns>

This is the starting tag of the **<Columns>** element. It is used to bind columns into the **GridView** control.

Line 23

<asp:BoundField DataField=“empId” HeaderText=“Emp ID”ReadOnly=“true” SortExpression=“empId” />

This line binds columns with their datafields. The **DataField=“empId”** attribute indicates that **empId** is the name of the field in the database to be bound. The **HeaderText=“Emp ID”** attribute indicates that **Emp ID** will be the header text for the **empId** field at runtime. The **ReadOnly=“true”** attribute indicates that the **empId** field will be read only. The **SortExpression=“empId”** attribute indicates that the sorting of data will be done according to the **empId** field.

Lines 24 and 25

These lines are similar to Line 23.

Line 26

</Columns>

This line indicates the end tag of the **<Columns>** element.

Line 27

</asp:GridView>

This line indicates the end tag of the **GridView** control.

Line 28

</ContentTemplate>

This line indicates the end tag of the **<ContentTemplate>** element.

Line 29

</asp:UpdatePanel>

This line indicates the end tag of the **UpdatePanel** control.

Line 30

**<asp:SqlDataSource ID=“SqlDataSource1” runat=“server”
ConnectionString= “< % \$ ConnectionStrings:Connect %>”
ProviderName= “< % \$ ConnectionStrings:Connect.ProviderName %>”
SelectCommand= “select empId, empName, empSalary from employee where (empName
like “ + @empName + “%)”>**

This line adds the data connectivity control, **SqlDataSource** to the form. This control is used to connect to the data source. Here, **SqlDataSource1** is the name of the control. The **runat="server"** attribute indicates that the control will execute at server-side. The **ConnectionString** attribute indicates the string, which is used to connect with the data source. The **ProviderName** indicates the name of the data provider that is used to connect with the data source. The **SelectCommand** attributes returns a query that will retrieve all those records from the **employee** table, whose name starts with the letter typed by the user.

Line 31

```
<SelectParameters>
```

This is the start tag of **<SelectParameters>**.

Line 32

```
<asp:ControlParameter ControlID="TextBox1" Name="empName" PropertyName="Text" Type="string" />
```

This line indicates the starting and closing of **ControlParameter** tag. The **ControlID** attribute indicates the name of the control to which you want to pass the value. The **Name** attribute indicates the actual name of the field in database from which you want to match the value. The **PropertyName** attribute indicates the property of the control. The **Type** attribute indicates the data type of the value.

Line 33

```
</SelectParameters>
```

This line indicates the end tag of **<SelectParameters>**.

Line 34

```
</asp:SqlDataSource>
```

This line indicates the end tag of the **SqlDataSource** control.

Line 35

```
</div>
```

This line indicates the end tag of the **<div>** element.

Line 36

```
</form>
```

This line indicates the end tag of the **<form>** element.

LIFE CYCLE OF PAGE REQUEST

The Page Request life cycle starts from the **ScriptManager** control. Basically, there is no major difference between the life cycle of an AJAX page and the life cycle of an ASP.NET page. The page execution life cycle is not altered even if the application uses partial rendering feature. In other words, the Page Request life cycle will remain the same even when you are using the **UpdatePanel** control. Therefore, in this case also, when an update occurs, the overhead on the server, as well as, the traffic between browser and server will remain intact. The **ScriptManager** control participates in the life cycle to facilitate partial page updates. The class, which manages partial page rendering in the browser, is called the **PageRequestManager** class.

The **PageRequestManager** class provides five events for asynchronous requests, and these are given in Table 2-4 below:

Event	Description
initializeRequest	This event is fired when the control is triggered for asynchronous partial postback by the user. You can add event handlers to this event.
beginRequest	This event is fired after the initializeRequest event and before the asynchronous postback is processed.
pageLoading	This is the first event that occurs after the server process is finished. The pageLoading event is fired after the page rendering event of the server.
pageLoaded	This event is fired after whole page is refreshed, whether it is synchronous or asynchronous postback.
endRequest	This is the last event of the Page Request Life Cycle. This event can be used to trace errors.

Table 2-4 The events of *PageRequestManager* class

Self-Evaluation Test

Answer the following questions and then compare them to those given at the end of this chapter:

- _____ is the process of reloading the entire page.
- The **UpdatePanel** control is located in the _____ namespace.
- _____ is the property of the **UpdatePanel** control, which defines the area that can be updated asynchronously.
- The content pages of the _____ page can be updated with the help of the **UpdatePanel** control.
- The _____ control shows the progress of refreshing the **UpdatePanel** control.
- The default value of the **ChildrenAsTriggers** property is **True**. (T/F)
- DisplayAfter** is the property of the **UpdatePanel** control. (T/F)
- The **ChildrenAsTriggers** property returns a boolean value. (T/F)
- Which of the following is not a property of the **UpdatePanel** control?
 - Triggers**
 - UpdateMode**
 - Comparison**
 - RenderMode**
- Which of the following is not a property of the **UpdateProgress** control?
 - DisplayAfter**
 - Visible**
 - AssociatedUpdatePanel**
 - UpdateMode**

Review Questions

Answer the following questions:

- Using the **UpdatePanel** control, you can update portions of a page synchronously. (T/F)
- Using the **UpdatePanel** control, you can restrict the flickering of a page. (T/F)
- RenderMode** is a property of the **UpdatePanel** control. (T/F)
- The **ScriptManager** control should be present on all pages, in which you want to apply partial page updates. (T/F)
- The **runat="server"** attribute indicates that the control will execute at server-side. (T/F)

6. In which of the following conditions will the **UpdatePanel** control always refresh?
- (a) When **UpdateMode** is **Always** and **ChildrenAsTriggers** is False.
 - (b) When **UpdateMode** is **Always** and **ChildrenAsTriggers** is True.
 - (c) When **UpdateMode** is **Conditional** and **ChildrenAsTriggers** is False.
 - (d) When **UpdateMode** is **Conditional** and **ChildrenAsTriggers** is True.
7. With which of the following controls, the **UpdatePanel** control will work properly?
- (a) **Menu**
 - (b) **TreeView**
 - (c) **GridView**
 - (d) **FileUpload**
8. For which of the following purposes, do you use the **ContentTemplate** property of the **UpdatePanel** control?
- (a) To define a template, which shows the contents of the **UpdatePanel** control.
 - (b) To define a template inside the **UpdatePanel** control that will be refreshed at runtime.
 - (c) To define a template outside the **UpdatePanel** control that will be refreshed at runtime.
 - (d) To define a template, which shows the properties of the **UpdatePanel** control.
9. For which of the following purposes, do you use the **ProgressTemplate** property of the **UpdateProgress** control?
- (a) To define a template, which shows the progress of refreshing the **UpdateProgress** control.
 - (b) To define a template, which shows the progress of refreshing the **UpdatePanel** control.
 - (c) To define a template, which shows the action performed by the **UpdateProgress** control.
 - (d) To define a template, which shows the action performed by the **UpdatePanel** control.
10. Which of the following is not an event of the **PageRequestManager** class?
- (a) **initializeRequest**
 - (b) **pageLoading**
 - (c) **endRequest**
 - (d) **pageRequest**

Exercises

Exercise 1

Create an application to update three different portions of a web page using the **UpdatePanel** controls.

Exercise 2

Create an application to show the progress of refreshing a web page using the **UpdateProgress** control.

Evaluation Copy. Do not reproduce. For information visit www.cadcam.com

Answers to Self-Evaluation Test

1. Postback, 2. System.Web.UI, 3. ContentTemplate, 4. Master, 5. UpdateProgress, 6. T, 7. F, 8. T, 9. c, 10. d.