ASP.NET 4.5

Lesson 10:
Introduction to ASP.NET AJAX

Lesson Objectives

- In this lesson, you will learn:
  - What is AJAX?
  - Advantages of AJAX
  - Technologies that makeup AJAX
  - How AJAX works?
  - What is ASP.NET AJAX?
  - ASP.NET development and how it led to AJAX
  - ASP.NET AJAX Server Controls and their use
    - ScriptManager Control
    - UpdatePanel Control
    - UpdateProgress Control
    - Timer Control

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

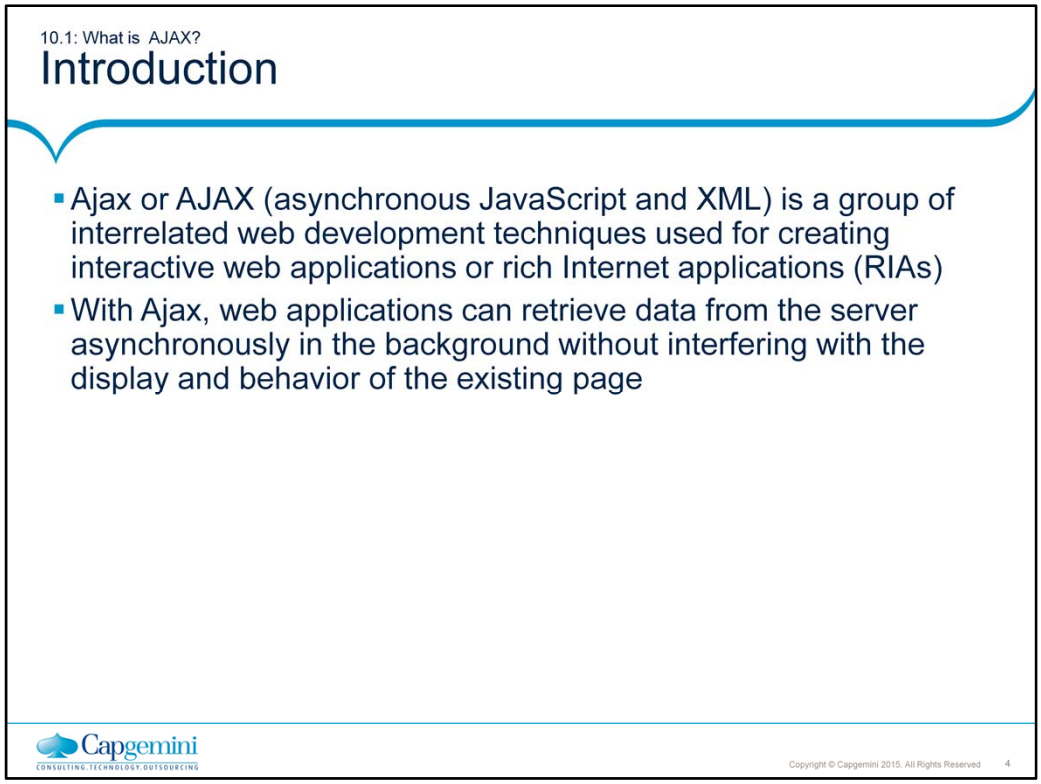
 2

Lesson Objectives

- In this lesson, you will learn (contd.):
  - ASP.NET AJAX Server Controls and their use
    - ScriptManager Control
    - UpdatePanel Control
    - UpdateProgress Control
    - Timer Control

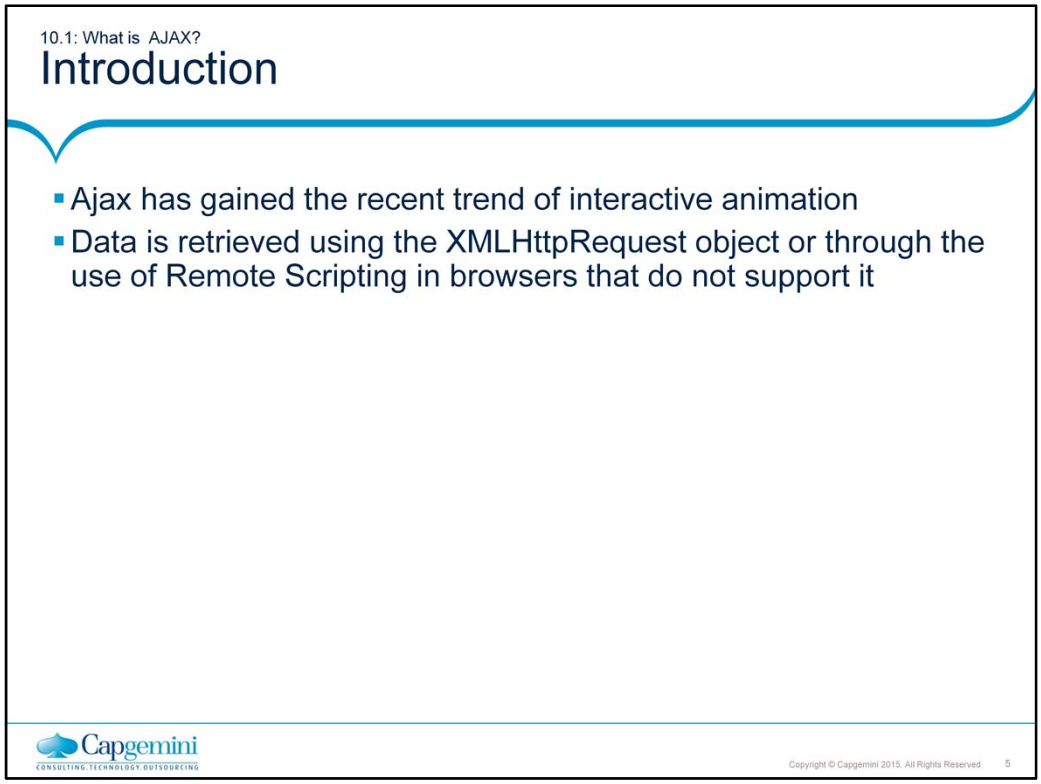
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3



What is AJAX?

- Ajax, or AJAX (asynchronous JavaScript and XML), is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications (RIAs). With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
- Ajax helps you in making your web application more interactive by retrieving small amount of data from web server and then showing it on your application. You can do all these things without refreshing your page.
- Usually in all the web applications, the user enters the data into the **form** and then clicks the **submit** button to submit the request to the server. Server processes the request and returns the view in a new page (by reloading the whole page). This process is inefficient, time consuming, and a little frustrating for the user if only a small amount of data exchange is required. For example, in an user registration form, it can be frustrating to the user when a whole page is reloaded only to check the availability of the user name. Ajax helps in making your application more interactive. With the help of Ajax you can tune your application to check the availability of the user name without refreshing the whole page.

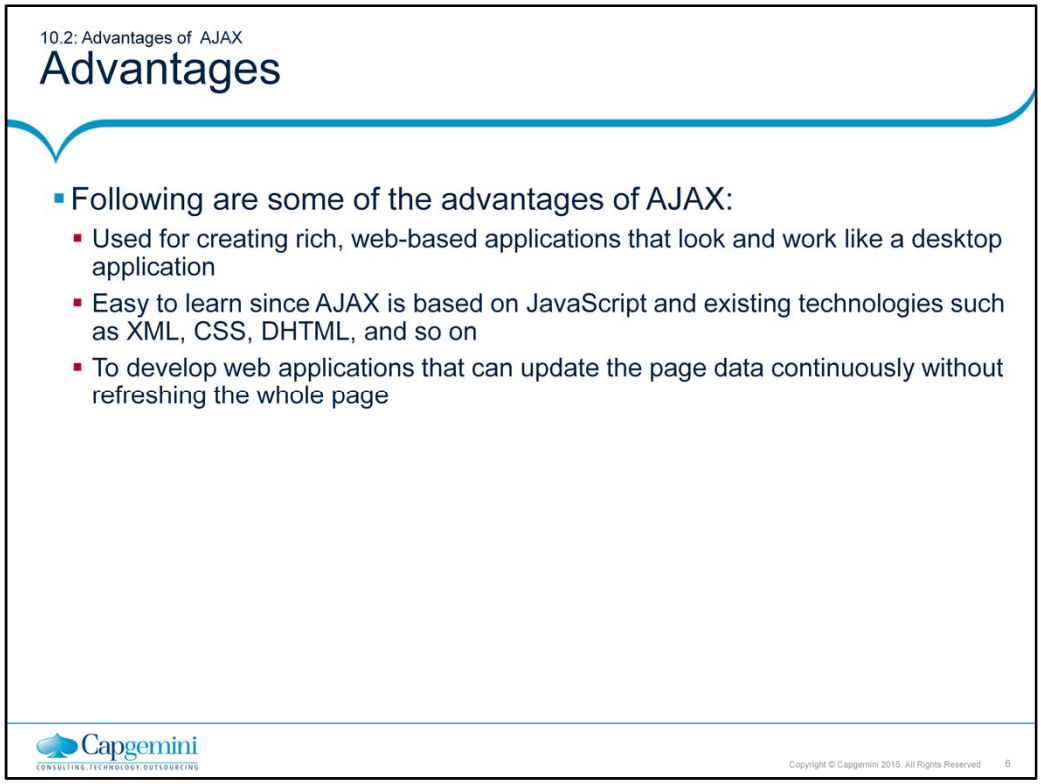


What is AJAX (contd.)?

Ajax has gained the recent trend of interactive animation.

Data is retrieved using the **XMLHttpRequest** object or through the use of Remote Scripting in browsers that do not support it.

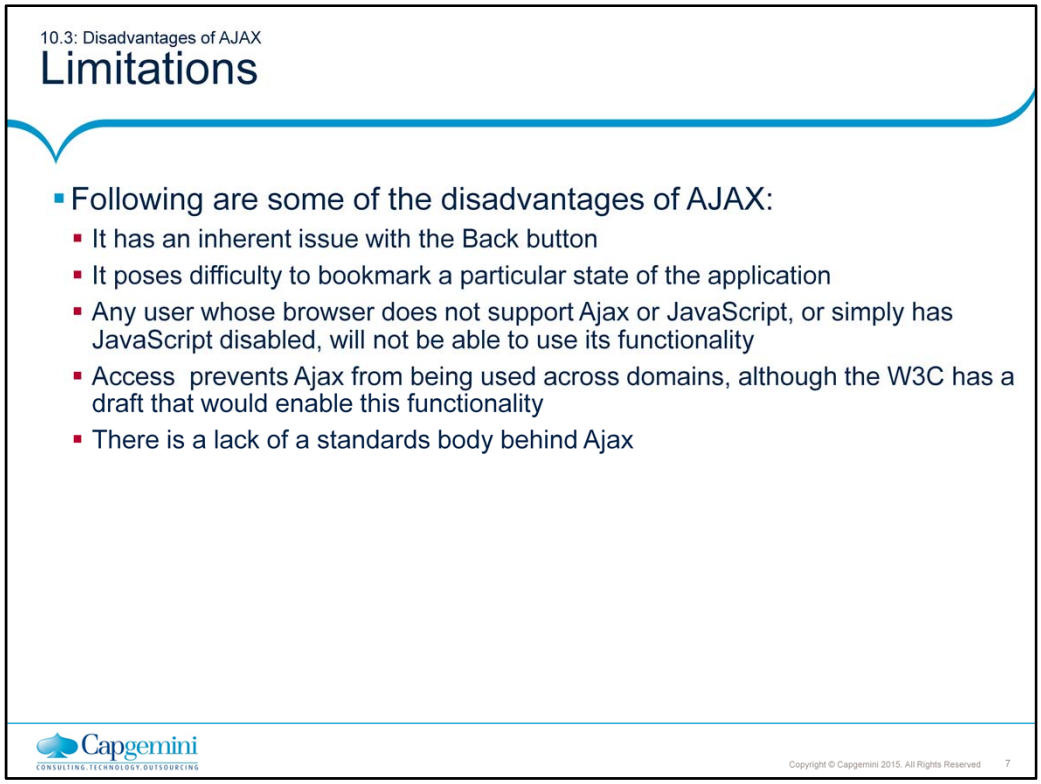
Despite the name, the use of JavaScript and XML is not required, and they do not have to be used asynchronously.



Advantages of AJAX:

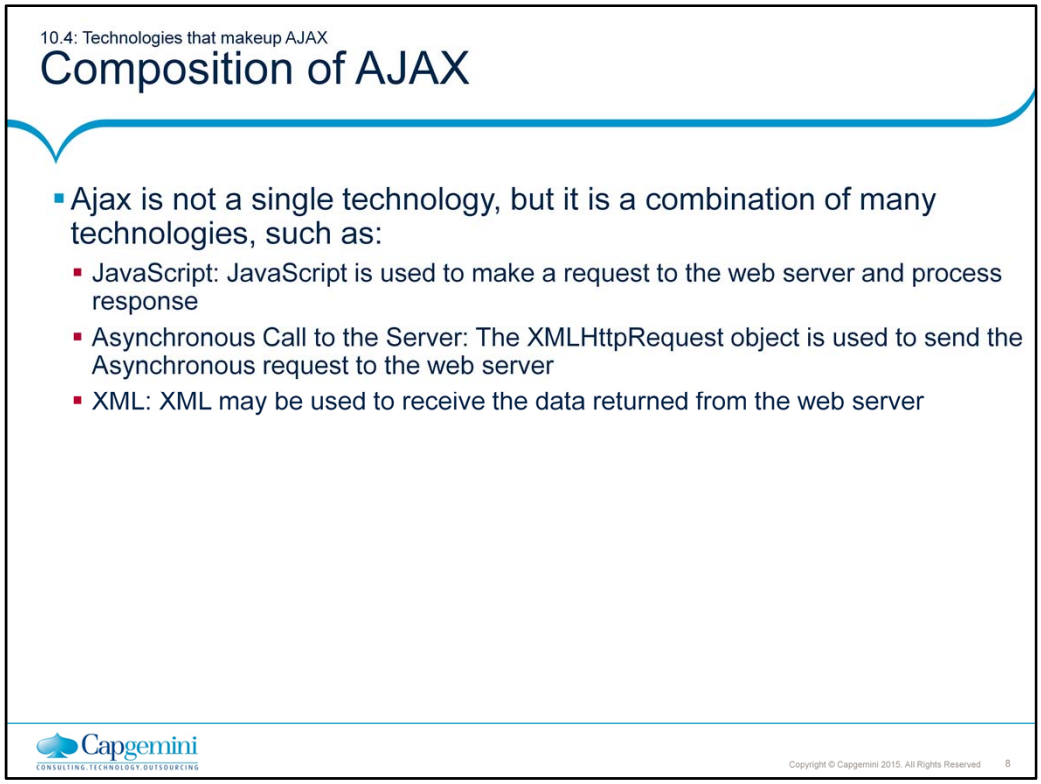
Ajax is a new very promising technology, which has become extremely popular these days. Here are the benefits of using Ajax:

- Ajax can be used for creating rich, web-based applications that look and work like a desktop application.
- Ajax is easy to learn. Ajax is based on JavaScript and existing technologies such as XML, CSS, DHTML, and so on. So it is very easy to learn Ajax.
- Ajax can be used to develop web applications that can update the page data continuously without refreshing the whole page.



Disadvantages of AJAX:

- Dynamically created pages do not register themselves with the browser's history engine, so clicking the browser's "**Back**" button does not return the user to an earlier state of the Ajax-enabled page. However, it instead returns them to the last page visited before it. Workarounds include the use of invisible **IFrames** to trigger changes in the browser's history and changing the anchor portion of the URL (following a #) when AJAX is run and monitoring it for changes.
- Dynamic web page updates also make it difficult for a user to bookmark a particular state of the application. Solutions to this problem exist, many of which use the URL fragment identifier (the portion of a URL after the '#') to keep track of, and allow users to return to, the application in a given state.
- Since most web crawlers do not execute JavaScript code, web applications should provide an alternative means of accessing the content that would normally be retrieved with Ajax, to allow search engines to index it.
- Any user whose browser does not support Ajax or JavaScript, or simply has JavaScript disabled, will not be able to use its functionality. Similarly, devices such as mobile phones, PDAs, and screen readers may not have support for JavaScript or the *XMLHttpRequest* object. Also, screen readers that are able to use Ajax may still not be able to properly read the dynamically generated content.
- The same origin policy prevents Ajax from being used across domains, although the W3C has a draft that would enable this functionality.
- Testing tools for Ajax often do not understand Ajax event models, data models, and protocols.
- Ajax inadvertently also opens up another attack vector for hackers that web developers might not fully test for.



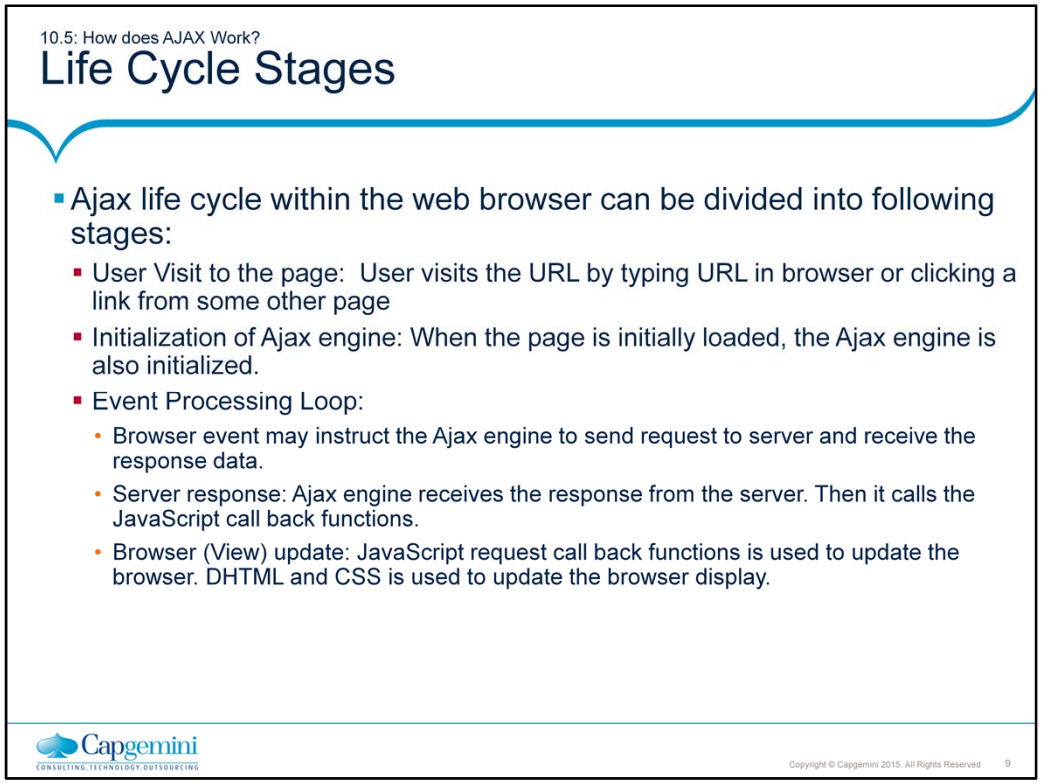
Technologies that makeup AJAX:

Ajax is not a single technology, but it is a combination of many technologies. These technologies are supported by modern web browsers. Following are techniques used in the Ajax applications.

- **JavaScript:** JavaScript is used to make a request to the web server. Once the response is returned by the webserver, more JavaScript can be used to update the current page. DHTML and CSS is used to show the output to the user. JavaScript is used very heavily to provide the dynamic behavior to the application.
- **Asynchronous Call to the Server:** Most of the Ajax applications use the XMLHttpRequest object to send the request to the web server. These calls are Asynchronous and there is no need to wait for the response to come back. User can do the normal work without any problem.
- **XML:** XML may be used to receive the data returned from the web server. JavaScript can be used to process the XML data returned from the web server easily.

Since then, however, there have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax. In particular, it has been noted that:

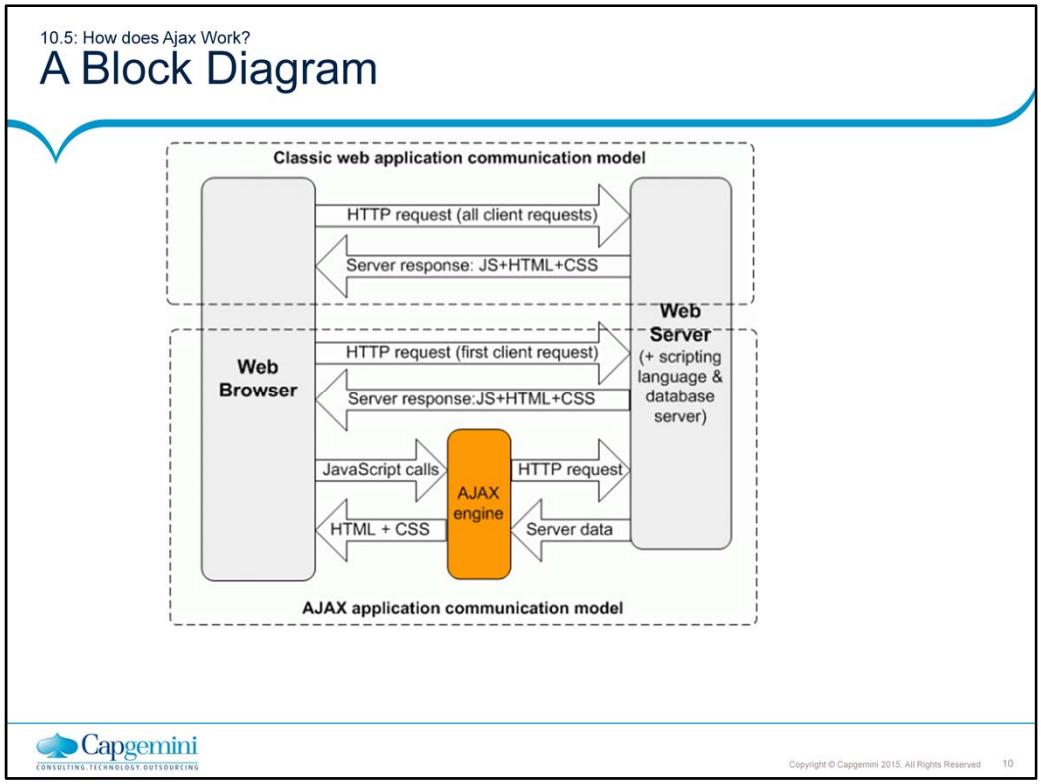
- JavaScript is not the only client-side scripting language that can be used for implementing an Ajax application. Other languages such as VBScript are also capable of the required functionality.
- XML is not required for data interchange and therefore XSLT is not required for the manipulation of data. The **JavaScript Notation object** (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used.



How does AJAX Work?

When user first visits the page, the Ajax engine is initialized and loaded. From that point of time the user interacts with Ajax engine to interact with the web server. The Ajax engine operates asynchronously while sending the request to the server and receiving the response from server. Ajax life cycle within the web browser can be divided into following stages:

1. **User Visit to the page:** User visits the URL by typing URL in browser or clicking a link from some other page.
2. **Initialization of Ajax engine:** When the page is initially loaded, the Ajax engine is also initialized. The Ajax engine can also be set to continuously refresh the page content without refreshing the whole page.
3. **Event Processing Loop:**
   - Browser event may instruct the Ajax engine to send request to server and receive the response data.
   - **Server response:** Ajax engine receives the response from the server. Then it calls the JavaScript call back functions.
   - **Browser (View) update:** JavaScript request call back functions is used to update the browser. DHTML and CSS is used to update the browser display.



Working of AJAX based Web Application:

- When an application uses AJAX, a new layer is added to the communication model. In the classic web application, the communication between the **client (the browser)** and the **web server** were performed directly, using HTTP requests.
- When the visitor requests a page, the server sends the full HTML and CSS code at once. After the visitor fills in a form and submits it, the server processes the information and rebuilds the page. It then sends the full page back to the client, and so on.
- While using AJAX, the page is loaded entirely only once, that is the first time it is requested. Besides the HTML and CSS code that make up the page, some JavaScript files are also downloaded, namely the **AJAX engine**. All requests for data to the sever will then be sent as JavaScript calls to this AJAX engine. The AJAX engine then requests information from the web server asynchronously. Thus only small **page bits** are requested and sent to the browser, as they are needed by the user. The engine then displays the information without reloading the entire page. This leads to a much more responsive interface. This is because only the necessary information is passed between the client and server, and not the whole page. This produces the feeling that information is displayed immediately, which brings web applications closer to their desktop relatives.
- At the heart of the AJAX method of communication with the server lies the **AJAX engine**. This is nothing more than some **JavaScript code** that instantiates and uses the **XMLHttpRequest object**. This is a JavaScript object that allows sending, receiving and processing HTTP requests to and from the server without refreshing the entire page.
- In **AJAX-powered applications**, HTTP requests for data can be made completely in the background, without the user experiencing any interruptions. This means the user can continue working and using the application, while the necessary page sections are received from the server. The XMLHttpRequest object was implemented as an ActiveX object in Internet Explorer, and has later become a native JavaScript object in most modern browsers (FireFox, Safari).
- Although adding an extra layer to any kind of model should add to the response time, this is an exception. Through the use of this new layer – the AJAX engine – response time shortens and the user interface seems much more connected to the application logic. Moreover, the user no longer has to wait around for the page to load.

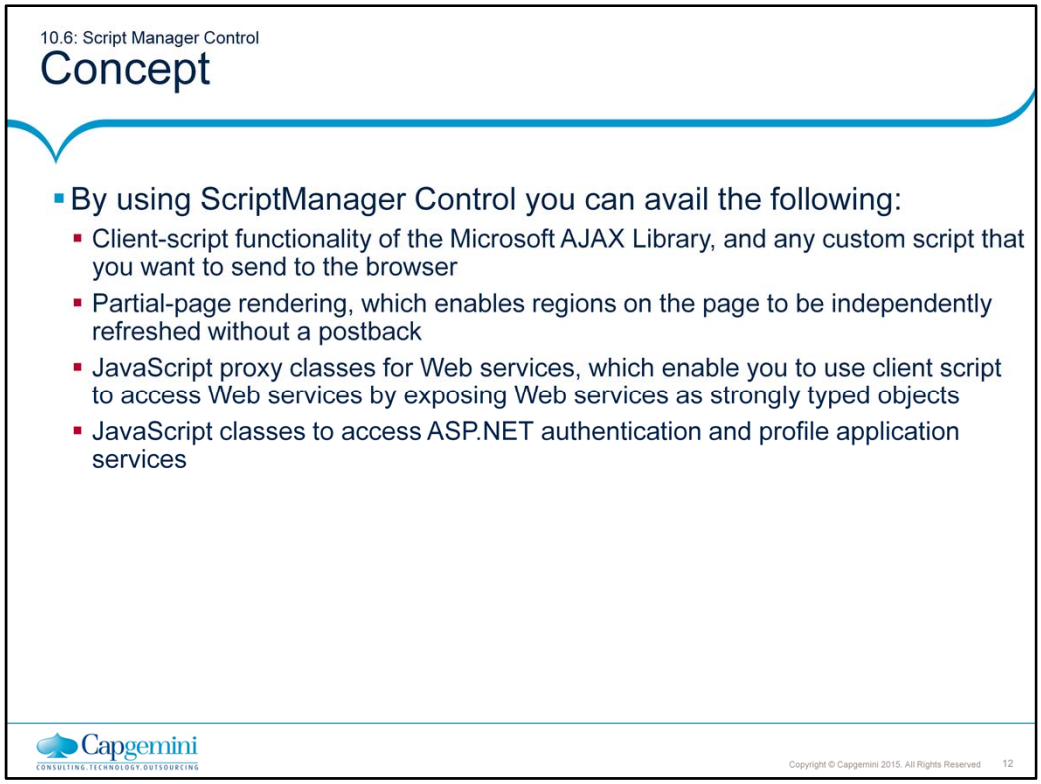
10.6: ScriptManager Control

Concept

- The ScriptManager control manages client script for Microsoft ASP.NET AJAX pages
  - By default, the ScriptManager control registers the script for the Microsoft AJAX Library with the page
  - This enables client script to use the type system extensions and to support features such as partial-page rendering and Web-service calls

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11



ScriptManager Control:

Why Use the ScriptManager Control?

You must use a ScriptManager control on a page to enable the following features of ASP.NET AJAX:

- Client-script functionality of the Microsoft AJAX Library, and any custom script that you want to send to the browser
  For more information, see ASP.NET AJAX and JavaScript.
- Partial-page rendering, which enables regions on the page to be independently refreshed without a postback
  The ASP.NET AJAX UpdatePanel, UpdateProgress, and Timer controls require a ScriptManager control to support partial-page rendering.
- JavaScript proxy classes for Web services
  They enable you to use client script to access Web services by exposing Web services as strongly typed objects.
- JavaScript classes to access ASP.NET authentication and profile application services

10.7: Integrating Client Script into ASP.NET Web Applications

Client Script and ASP.NET Web Applications

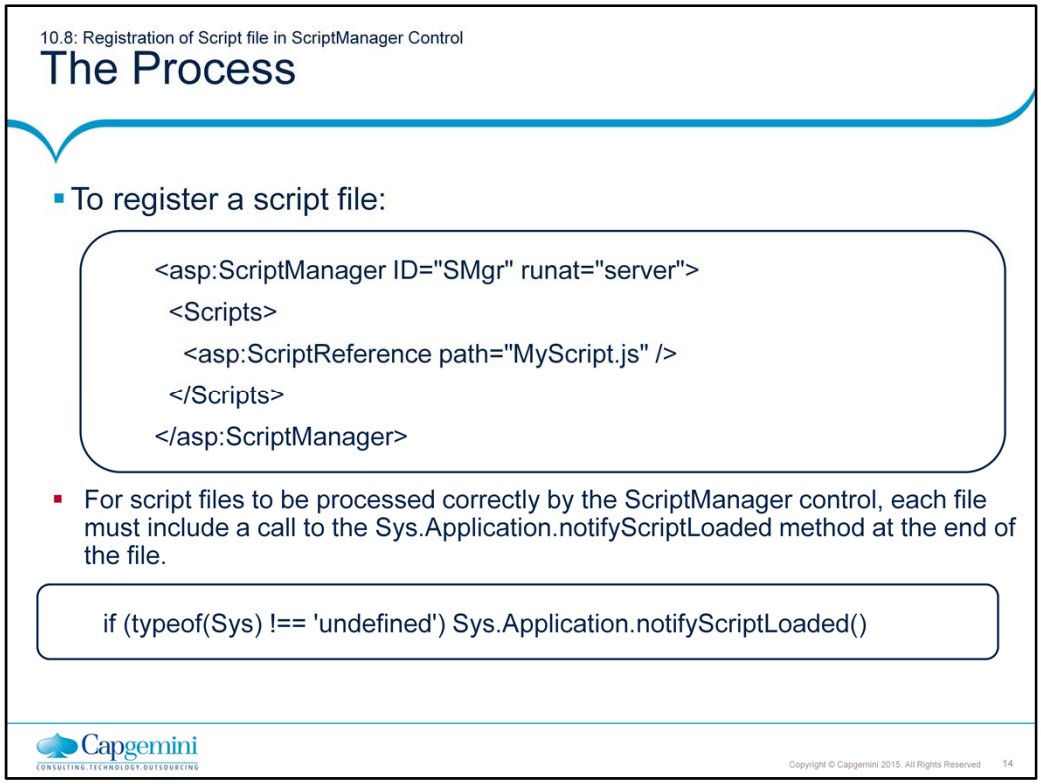
- Any ASP.NET Web page can access a script file by referring to it in a <script> block.

```
<script type="text/javascript" src="MyScript.js"></script>
```

  - However, a script invoked in this manner cannot participate in partial-page rendering or access certain components of the Microsoft AJAX Library
  - To make a script file available for partial-page rendering in an ASP.NET AJAX Web application, the script must be registered with the ScriptManager control on the page

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13



Registration of Script file in ScriptManager Control:

- To register a script file, create a ScriptReference object that points to the file question and that adds it to the Scripts collection.
- The following example shows how to do this in markup:

```
<asp:ScriptManager ID="SMgr" runat="server">
  <Scripts>
    <asp:ScriptReference path="MyScript.js" />
  </Scripts>
</asp:ScriptManager>
```

- For script files to be processed correctly by the ScriptManager control, each file must include a call to the **Sys.Application.notifyScriptLoaded** method at the end of the file. This call notifies the application that the file has finished loading. The following example shows the code to use for this purpose:

```
if (typeof(Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();
```

10.8: Registration of Script file in ScriptManager Control

The Process

- If your script is embedded in an assembly, then:
  - Notification statement need not be included in the script
  - No need to specify a path attribute in the script reference

```
<asp:ScriptManager ID="SMgr" runat="server">
  <Scripts>
    <asp:ScriptReference Name="MyScript.js"
Assembly="MyScriptAssembly"/>
  </Scripts>
</asp:ScriptManager>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Registration of Script file in ScriptManager Control:

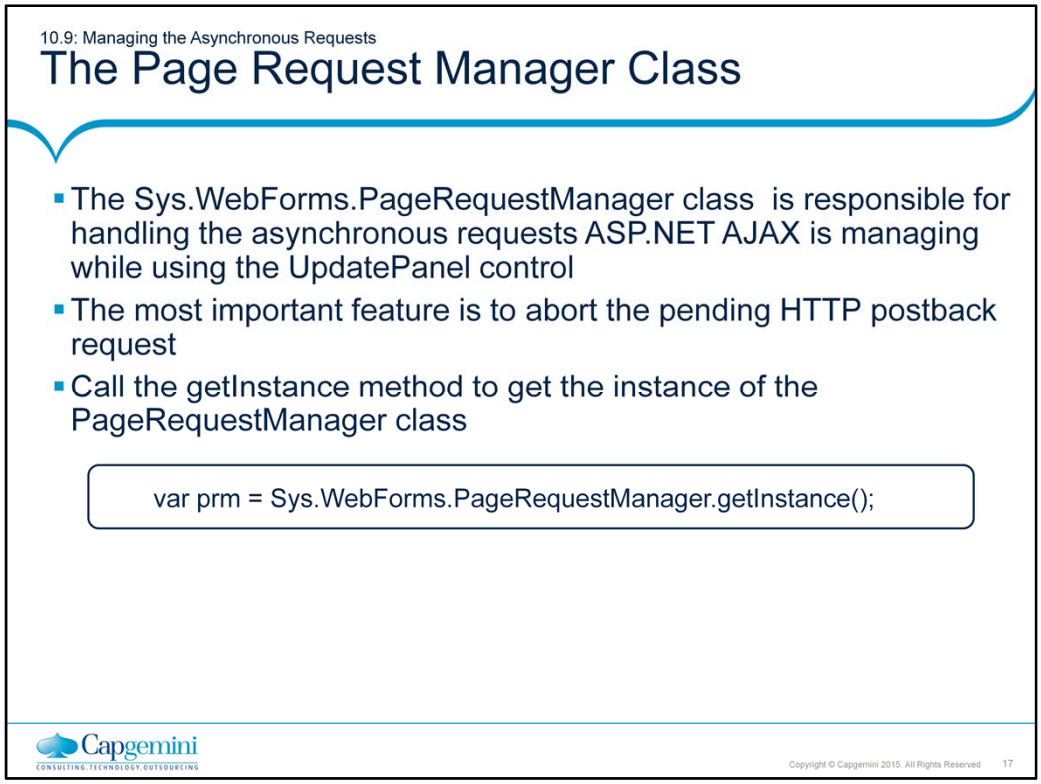
- If your script is embedded in an assembly, then you do not have to include a notification statement in the script. You also do not have to specify a path attribute in the script reference. However, you must provide the name of the assembly without the file name extension, as shown in the following example:

```
<asp:ScriptManager ID="SMgr" runat="server">
  <Scripts>
    <asp:ScriptReference
      Name="MyScript.js" Assembly="MyScriptAssembly"/>
  </Scripts>
</asp:ScriptManager>00
```

- **Note:** This scenario is not common for page developers, because most controls with embedded script libraries reference their scripts internally. For more information, see Embedding a JavaScript File as a Resource in an Assembly.
- You can also register scripts programmatically by creating script references in code and then adding them to the Scripts collection. You can register scripts that are required for partial-page updates by using the registration methods of the ScriptManager control.
- You can use these methods in the following ways:
  - To generate client script in code, build a block of script as a string and pass it to the RegisterClientScriptBlock) method.
  - To add standalone script files that have no Microsoft AJAX Library dependencies, use the RegisterClientScriptInclude) method.
  - To add script files that are embedded in an assembly, use the RegisterClientScriptResource method.

- **Note:** Scripts that are registered by using these methods do not have localization support.

- For a complete list of script-registration methods and their uses, see the ScriptManager control overview.
- Any script blocks or inline script that you are registering must be inside the page's **<form>** element. Otherwise, the script is not registered with the ScriptManager control and cannot access ASP.NET AJAX functionality. For more information, see **initialize** Method.

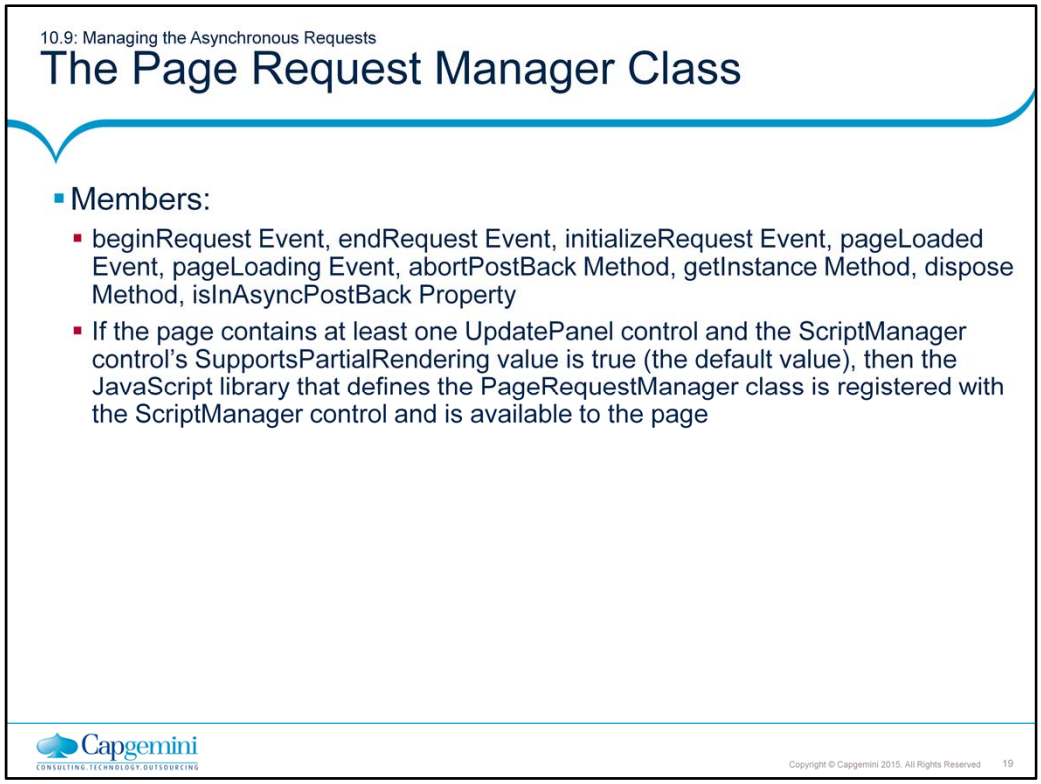


Managing the Asynchronous Requests (PageRequestManager Class):

- You do not create a new instance of the **PageRequestManager** class directly. Instead, an instance is available when partial-page rendering is enabled. Call the **getInstance** method to get the instance of the **PageRequestManager** class.

```
var prm = Sys.WebForms.PageRequestManager.getInstance();
```

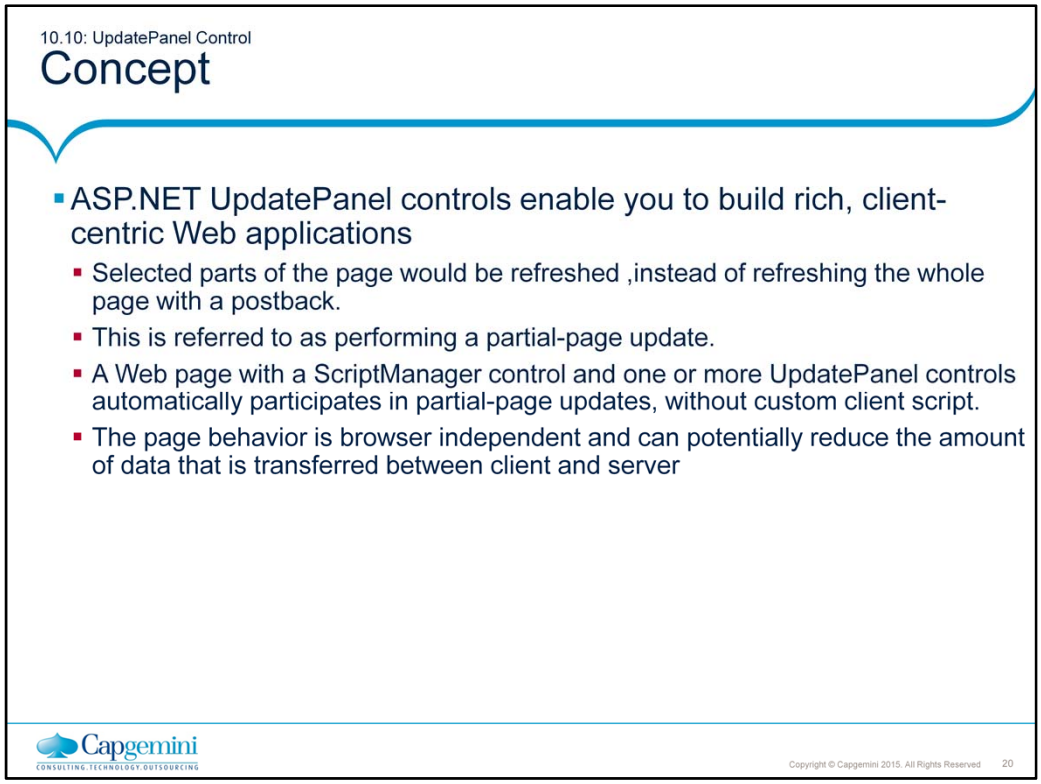
```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void ProcessClick_Handler(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(2000);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>PageRequestManager beginRequest Example</title>
    <style type="text/css">
    body { font-family: Tahoma;    }
    div.AlertStyle
    {
      background-color: #FFC080; top: 95%; left: 1%; height: 20px; width: 270px;
      position: absolute; visibility: hidden;
    }
   </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server" />
            <script type="text/javascript" language="javascript">

Sys.WebForms.PageRequestManager.getInstance().add_beginRequest(BeginRequestHandler);
              Sys.WebForms.PageRequestManager.getInstance().add_endRequest(EndRequestHandler);
               function BeginRequestHandler(sender, args) {
                  var elem = args.get_postBackElement();
                  ActivateAlertDiv('visible', 'AlertDiv', elem.value + ' processing...');
               }
               function EndRequestHandler(sender, args) {
                  ActivateAlertDiv('hidden', 'AlertDiv', '');
               }
               function ActivateAlertDiv(visstring, elem, msg) {
                  var adiv = $get(elem);
                  adiv.style.visibility = visstring;
                  adiv.innerHTML = msg;
               }
            </script>

            <asp:UpdatePanel ID="UpdatePanel1" UpdateMode="Conditional" runat="Server">
               <ContentTemplate>
                  <asp:Panel ID="Panel1" runat="server" GroupingText="Update Panel">
                     Last update:
                     <%= DateTime.Now.ToString()%>.
                     <br />
                     <asp:Button runat="server" ID="Button1" Text="Process 1"
OnClick="ProcessClick_Handler" />
                     <asp:Button runat="server" ID="Button2" Text="Process 2"
OnClick="ProcessClick_Handler" />
                  </asp:Panel>
               </ContentTemplate>
            </asp:UpdatePanel>
            <div id="AlertDiv" class="AlertStyle">
            </div>
        </div>
    </form>
</body>
</html>
```



The PageRequestManager Class:
Members:
Following are the members of the PageRequestManager Class:

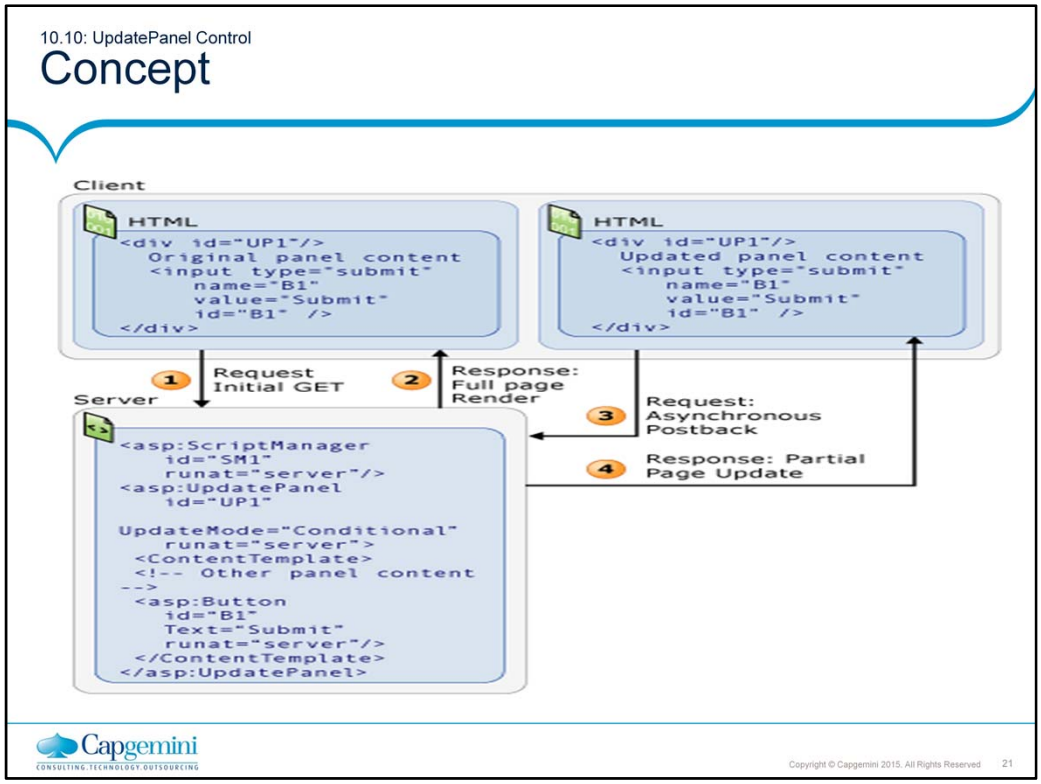
- **beginRequest Event:** It is raised before processing of an asynchronous postback starts and the postback request is sent to the server.
- **endRequest Event:** It is raised after an asynchronous postback is finished and control has been returned to the browser.
- **initializeRequest Event:** It is raised during the initialization of the asynchronous postback.
- **pageLoaded Event:** It is raised after all content on the page is refreshed as the result of either a synchronous or an asynchronous postback.
- **pageLoading Event:** It is raised after the response from the server to an asynchronous postback is received but before any content on the page is updated.
- **abortPostBack Method:** It stops all updates that would occur as a result of an asynchronous postback.
- **dispose Method:** It **r**eleases ECMAScript (JavaScript) resources and detaches events.
- **getInstance Method:** It returns the instance of the PageRequestManager class for the page.
- **isInAsyncPostBack Property:** It returns a value that indicates whether the PageRequestManager object is processing a postback.

Suppose the page contains at least one UpdatePanel control and the ScriptManager control's **SupportsPartialRendering** value is true (the default value). Then the JavaScript library that defines the PageRequestManager class is registered with the ScriptManager control and is available to the page.



UpdatePanel Control:

- The UpdatePanel control is a server control that helps you develop Web pages with complex client behavior that makes a Web page appear more interactive to the end user.
- Coordinating between **server** and **client** to update only specified parts of a Web page usually requires in-depth knowledge of **ECMAScript (JavaScript)**. However, by using the **UpdatePanel control**, you can enable a Web page to participate in partial-page updates without writing any client script.
- If you want, you can add **custom client script** to enhance the client user experience. When you use an UpdatePanel control, the page behavior is browser independent and can potentially reduce the amount of data that is transferred between client and server.



UpdatePanel Control:

- **UpdatePanel** controls work by specifying regions of a page that can be updated without refreshing the whole page. This process is coordinated by the **ScriptManager** server control and the client **PageRequestManager** class.
- When partial-page updates are enabled, controls can asynchronously post to the server.
  - An **asynchronous postback** behaves like a **regular postback** in that the resulting server page executes the complete page and control life cycle. However, with an **asynchronous postback**, page updates are limited to regions of the page that are enclosed in **UpdatePanel** controls and that are marked to be updated.
  - The server sends HTML markup for only the affected elements to the browser. In the browser, the client PageRequestManager class performs **Document Object Model (DOM)** manipulation to replace existing HTML with updated markup.
- The above illustration shows a page that is loaded for the first time, and a subsequent asynchronous postback that refreshes the content of an **UpdatePanel** control.

10.10: UpdatePanel Control

Concept

- Enabling Partial-Page Updates:
  - The UpdatePanel control requires a ScriptManager control in the Web page.
  - By default, partial-page updates are enabled because the default value of the EnablePartialRendering property of the ScriptManager control is true.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

10.10: UpdatePanel Control

Concept

- Specifying UpdatePanel Control Content:
  - You can add content to an UpdatePanel control declaratively or in the designer by using the **ContentTemplate** property
  - In markup, this property is exposed as a <ContentTemplate> element.
  - To add content programmatically, you use the **ContentTemplateContainer** property
  - When a page that contains one or more UpdatePanel controls is first rendered, all the contents of the UpdatePanel controls are rendered and sent to the browser
  - On subsequent asynchronous postbacks, the content of individual UpdatePanel controls might be updated
  - Updates depend on the panel settings, on what element caused the postback, and on code that is specific to each panel

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

10.10: UpdatePanel Control

Concept

- Specifying UpdatePanel Triggers
  - By default, any postback control inside an UpdatePanel control causes an **asynchronous postback** and refreshes the panel's content. However, you can also configure other controls on the page to refresh an UpdatePanel control. You do this by defining a trigger for the UpdatePanel control
  - **A trigger** is a binding that specifies the postback control and event that causes a panel to update. When the specified event of the trigger control is raised, the update panel is refreshed
  - The trigger is defined by using **<asp:AsyncPostBackTrigger>** element inside the **<Triggers>** element of the UpdatePanel control
  - A trigger's control event is optional. If you do not specify an event, then the trigger event is the default event of the control. For example, for the Button control, the default event is the Click event

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 24

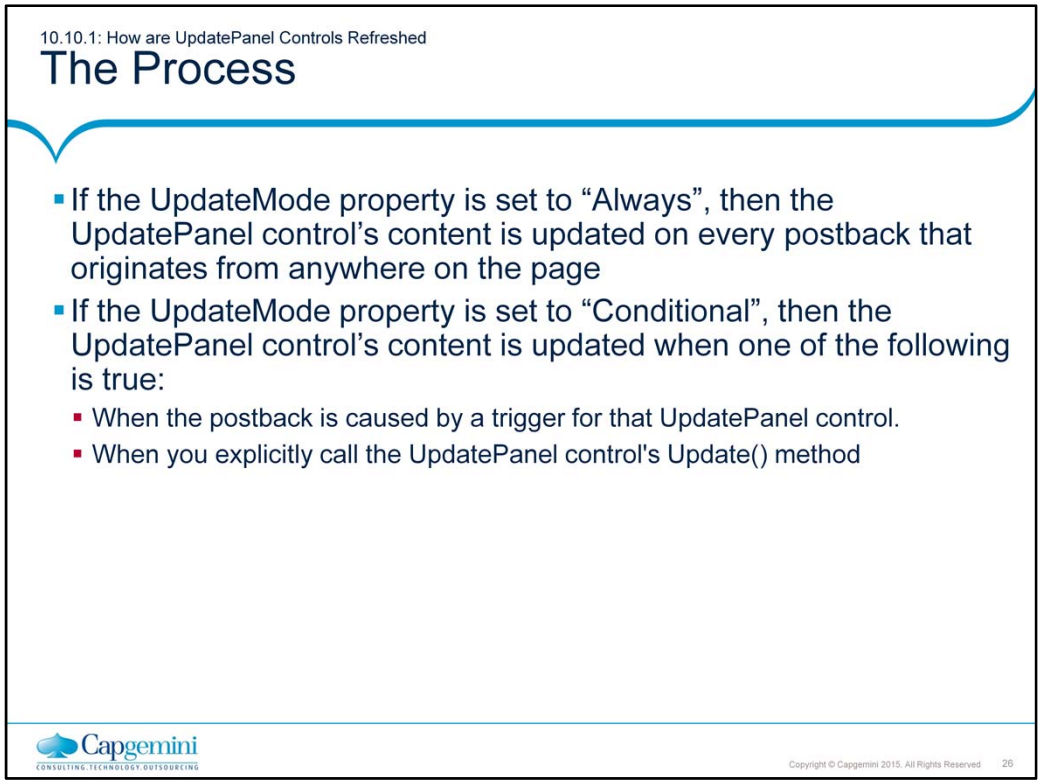
10.10: UpdatePanel Control

Concept

- Demo on UpdatePanel Control

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25



How are UpdatePanel Controls Refreshed?

- The following list describes the property settings of the UpdatePanel control that determine when a panel's content is updated during partial-page rendering.
- If the **UpdateMode** property is set to “**Always**”, then the UpdatePanel control’s content is updated on every postback that originates from anywhere on the page. This includes:
  - Asynchronous postbacks from controls that are inside other UpdatePanel controls
  - Postbacks from controls that are not inside UpdatePanel controls
- If the **UpdateMode** property is set to “**Conditional**”, then the UpdatePanel control’s content is updated when one of the following is true:
  - When the postback is caused by a trigger for that **UpdatePanel control**
  - When you explicitly call the UpdatePanel control’s **Update() method**
  - When the UpdatePanel control is nested inside another UpdatePanel control and the parent panel is updated
  - When the **ChildrenAsTriggers** property is set to true and any child control of the UpdatePanel control causes a postback. Child controls of nested UpdatePanel controls do not cause an update to the outer UpdatePanel control unless they are explicitly defined as triggers for the parent panel.
    - If the **ChildrenAsTriggers** property is set to “**false**” and the **UpdateMode** property is set to “**Always**”, then an exception is thrown. The **ChildrenAsTriggers** property is intended to be used only when the **UpdateMode** property is set to “**Conditional**”.

10.10.1: How are UpdatePanel Controls Refreshed

The Process

- When the UpdatePanel control is nested inside another UpdatePanel control and the parent panel is updated
- When the ChildrenAsTriggers property is set to true, Child controls of nested UpdatePanel controls do not cause an update to the outer UpdatePanel control unless they are explicitly defined as triggers for the parent panel
- The ChildrenAsTriggers property is intended to be used only when the UpdateMode property is set to “Conditional”

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 27

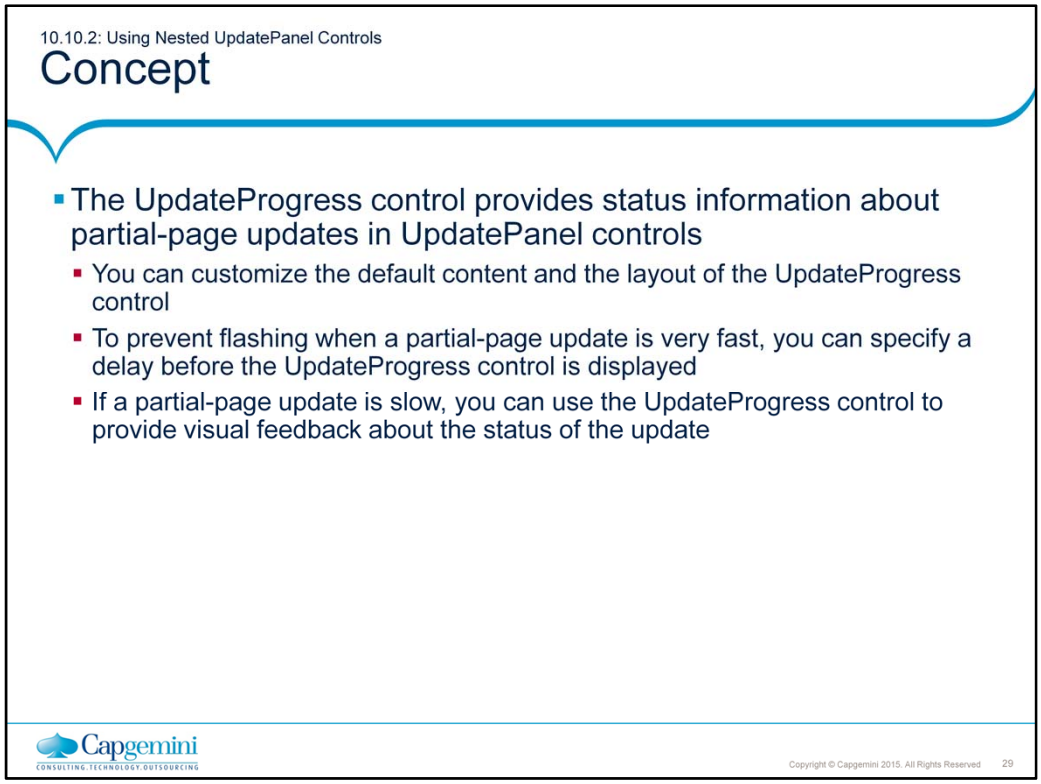
10.10.2: Using Nested UpdatePanel Controls

Concept

- UpdatePanel controls can be nested. If the parent panel is refreshed, then all nested panels are refreshed also
- Suppose markup defines an UpdatePanel control inside another UpdatePanel control
  - Then a button in the parent panel triggers an update of the content in both the parent and the child panel
  - The button in the child panel triggers an update of only the child panel

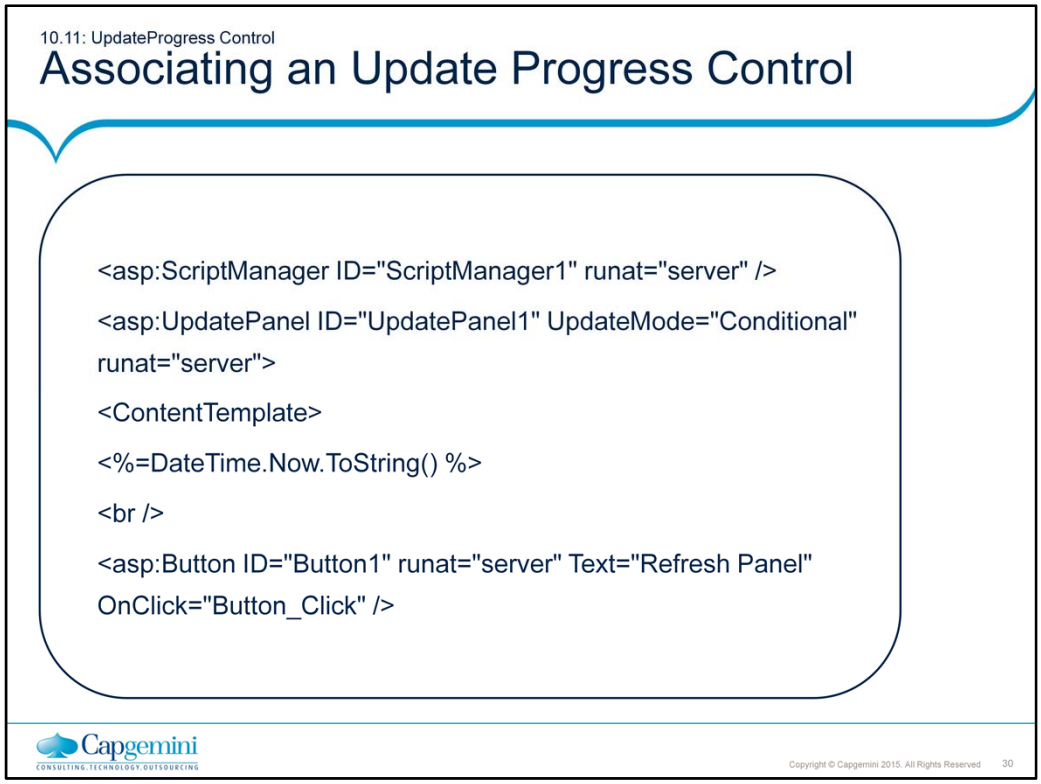
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28



UpdateProgress Control:

- The **UpdateProgress** control provides status information about partial-page updates in **UpdatePanel** controls. You can customize the default content and the layout of the **UpdateProgress** control. To prevent flashing when a partial-page update is very fast, you can specify a delay before the **UpdateProgress** control is displayed.
- The UpdateProgress control helps you design a more intuitive UI when a Web page contains one or more UpdatePanel controls for partial-page rendering. If a partial-page update is slow, you can use the **UpdateProgress** control to provide visual feedback about the status of the update.
  - You can put multiple **UpdateProgress** controls on a page, each associated with a different **UpdatePanel** control.
  - Alternatively, you can use one **UpdateProgress** control and associate it with all **UpdatePanel** controls on the page.
- The **UpdateProgress** control renders a ** ** element that is displayed or hidden depending on whether an associated **UpdatePanel** control has caused an asynchronous postback. For initial page rendering and for synchronous postbacks, the UpdateProgress control is not displayed.



UpdateProgress Control:

Associating an Update Progress Control with an UpdatePanel Control:

- You can associate an **UpdateProgress** control with an **UpdatePanel** control by setting the **AssociatedUpdatePanelID** property of the UpdateProgress control.
- When a postback event originates from an UpdatePanel control, any associated UpdateProgress controls are displayed. If you do not associate the **UpdateProgress** control with a specific **UpdatePanel** control, then the **UpdateProgress** control displays progress for any asynchronous postback.
- Suppose the **ChildrenAsTriggers** property of an **UpdatePanel** control is set to false and an asynchronous postback originates from inside that **UpdatePanel** control. Then any associated UpdateProgress controls will be displayed.

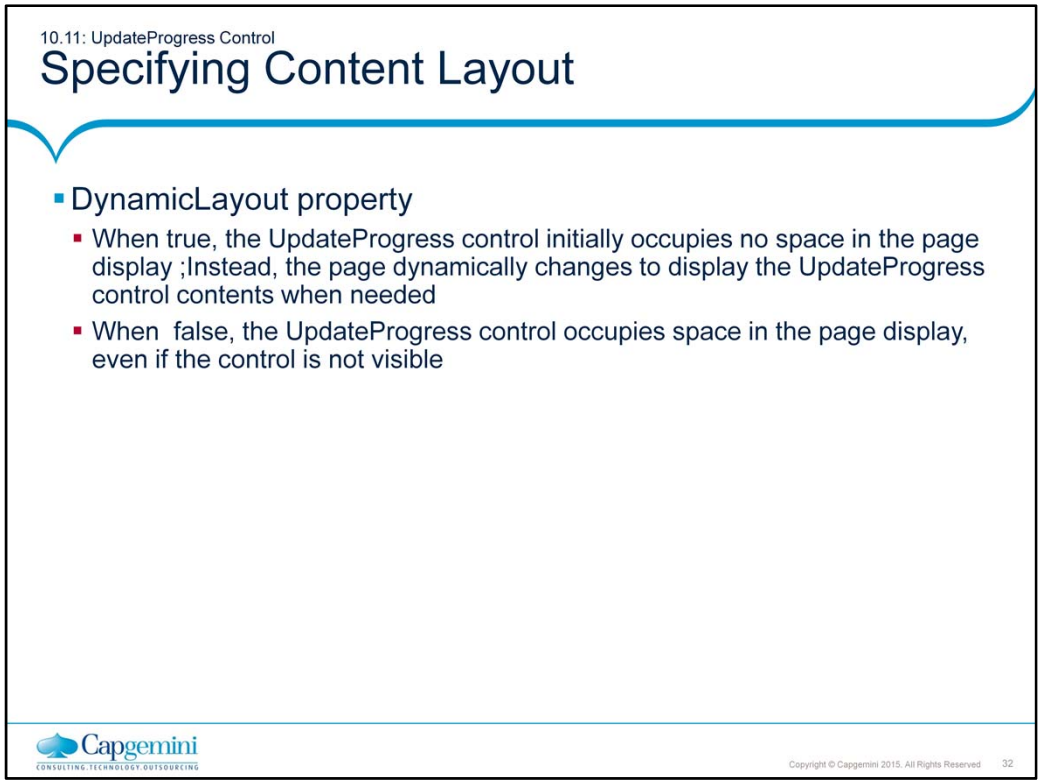
10.11: UpdateProgress Control

Associating an Update Progress Control

```
<asp:UpdateProgress ID="UpdateProgress1"
AssociatedUpdatePanelID="UpdatePanel1" runat="server">
<ProgressTemplate> UpdatePanel1 updating... </ProgressTemplate>
</asp:UpdateProgress>
 </ContentTemplate>
</asp:UpdatePanel>
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

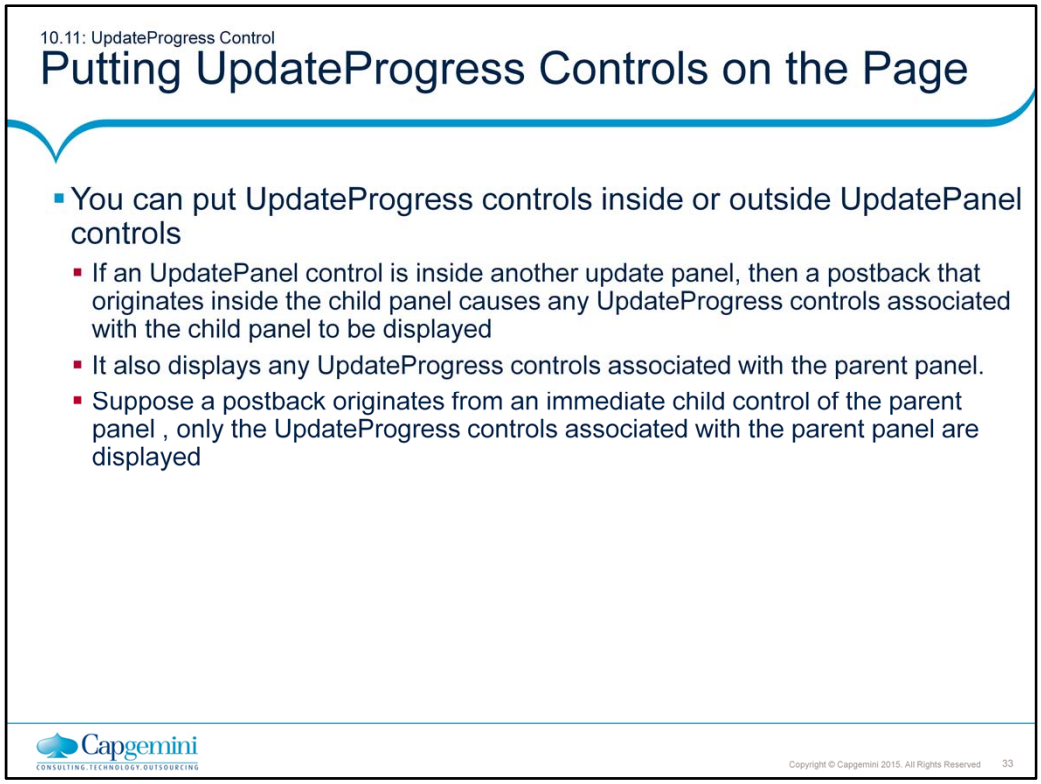
 31



UpdateProgress Control:

Specifying Content Layout:

- When the **DynamicLayout** property is *true*, the **UpdateProgress** control initially occupies no space in the page display.
  - Instead, the page dynamically changes to display the UpdateProgress control contents when needed.
  - To support dynamic display, the control is rendered as a ** ** element that has its display style property initially set to none.
- When the **DynamicLayout** property is *false*, the **UpdateProgress** control occupies space in the page display, even if the control is not visible.
  - In that case, the ** ** element for the control has its display **style** property set to block and its visibility initially set to hidden.



UpdateProgress Control:

Putting UpdateProgress Controls on the Page:

- You can put **UpdateProgress** controls inside or outside **UpdatePanel** controls. An UpdateProgress control is displayed whenever the UpdatePanel control it is associated with is updated as a result of an asynchronous postback. This is true even if the UpdateProgress control is inside another UpdatePanel control.
  - If an **UpdatePanel** control is inside another update panel, then a postback that originates inside the child panel causes any UpdateProgress controls associated with the child panel to be displayed.
  - It also displays any **UpdateProgress** controls associated with the parent panel. If a postback originates from an immediate child control of the parent panel, only the UpdateProgress controls associated with the parent panel are displayed. This follows the logic for how postbacks are triggered.

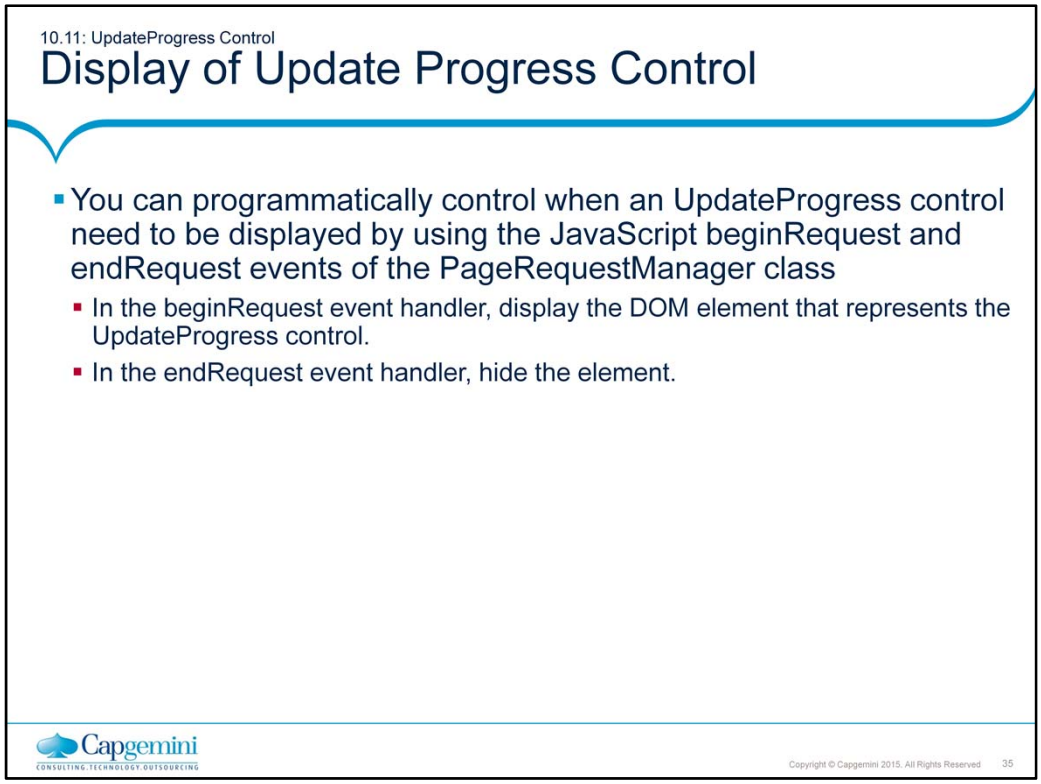
10.11: UpdateProgress Control

Demo

- Demo on using UpdateProgress Control

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34



UpdateProgress Control:

Specifying When UpdateProgress Controls Are Displayed:

- You can programmatically control when an **UpdateProgress** control to be displayed by using the JavaScript **beginRequest** and **endRequest** events of the PageRequestManager class.
  - In the **beginRequest** event handler, display the DOM element that represents the UpdateProgress control.
  - In the **endRequest** event handler, hide the element.
- You must provide client script to show and hide an **UpdateProgress** control in the following circumstances:
  - During a postback from a control that is registered as an asynchronous postback trigger for the update panel, but that the UpdateProgress control is not associated with
  - During postbacks from controls that are registered programmatically as asynchronous postback controls by using the **RegisterAsyncPostBackControl(Control)** method of the ScriptManager control. In that case, the UpdateProgress control cannot determine automatically that an asynchronous postback has been triggered.

10.12: Timer Control

Concept

- The Timer control performs postbacks at defined intervals.
- If you use the Timer control with an UpdatePanel control, then you can enable partial-page updates at a defined interval
  - You can also use Timer control to post the whole page

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

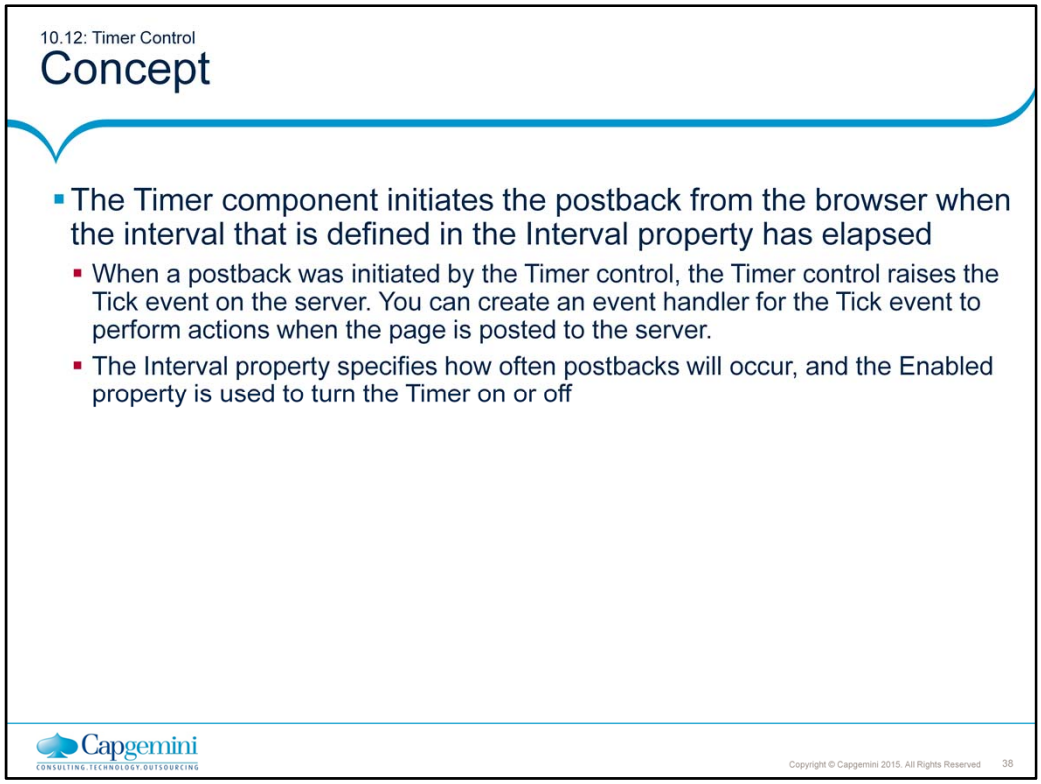
10.12: Timer Control

Concept

- You use the Timer control when you want to do the following tasks:
  - Periodically update the contents of one or more UpdatePanel controls without refreshing the whole Web page
  - Run code on the server every time that a Timer control causes a postback
  - Synchronously post the whole Web page to the Web server at defined intervals.

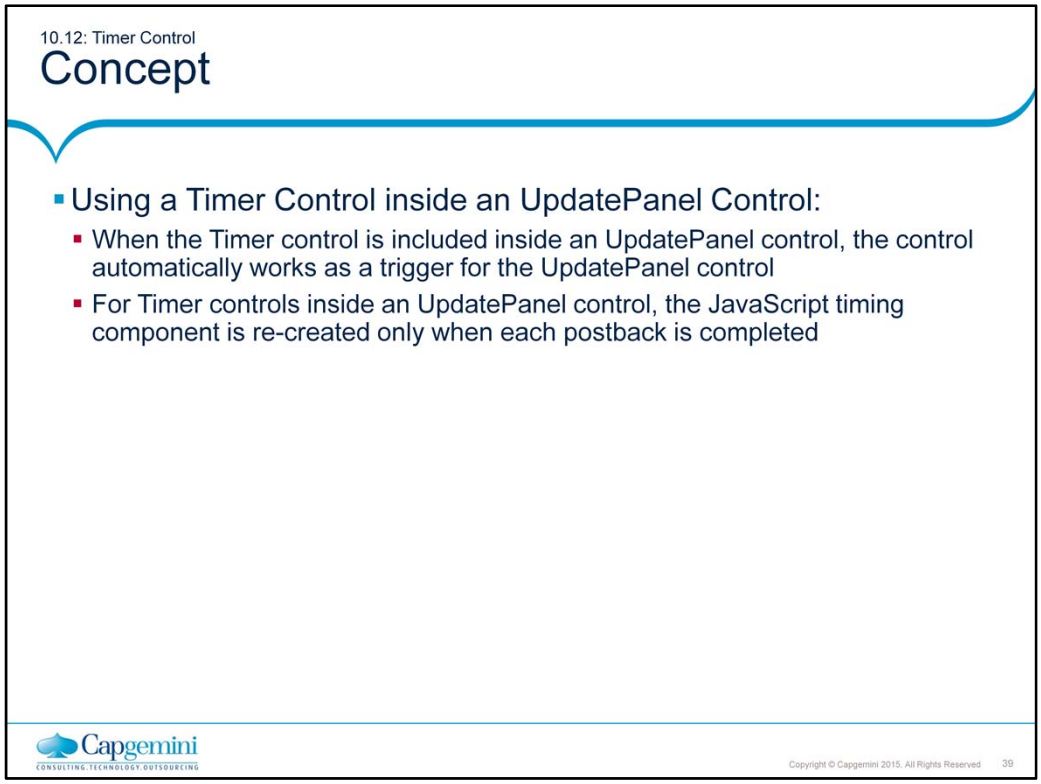
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 37



Timer Control:

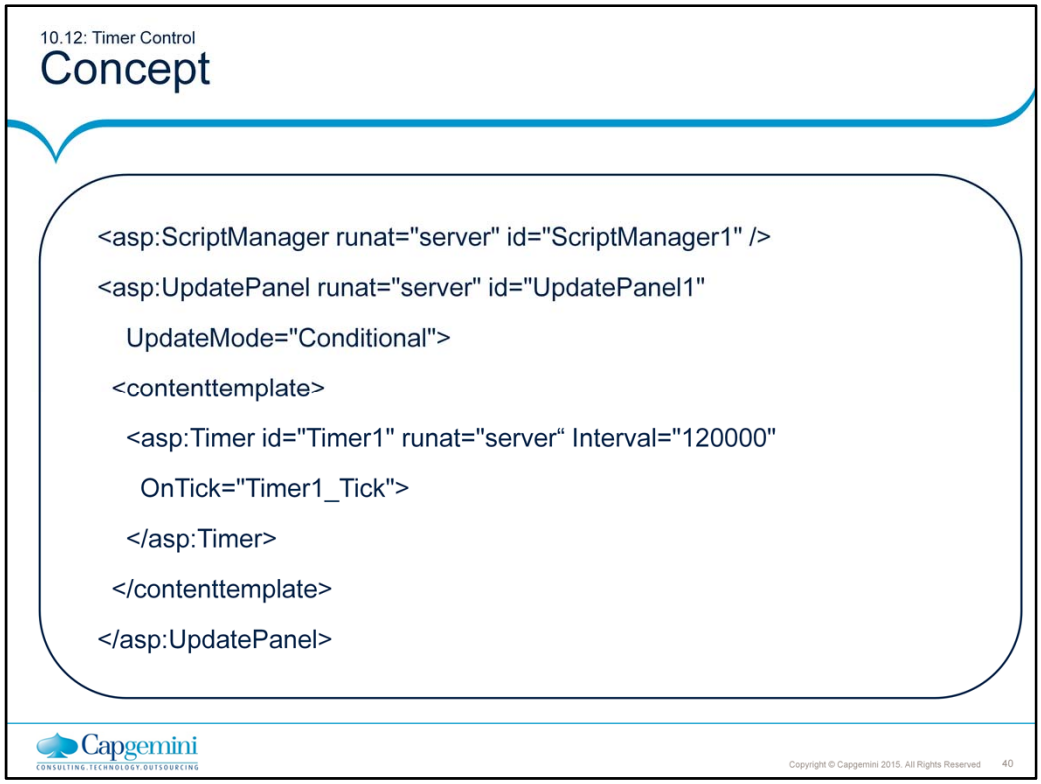
- The **Timer** control is a server control that embeds a JavaScript component into the Web page. The JavaScript component initiates the postback from the browser when the interval that is defined in the **Interval** property has elapsed. You set the properties for the **Timer** control in code that runs on the server and those properties are passed to the **JavaScript** component.
- An instance of the **ScriptManager** class must be included in the Web page when you use the **Timer** control.
    - When a postback is initiated by the Timer control, the Timer control raises the **Tick** event on the server. You can create an event handler for the **Tick** event to perform actions when the page is posted to the server.
    - Set the **Interval** property to specify how often postbacks will occur, and set the **Enabled** property to turn the Timer on or off. The Interval property is defined in milliseconds and has a default value of 60,000 milliseconds, or 60 seconds.
    - **Note:** Setting the Interval property of a Timer control to a small value can generate significant traffic to the Web server. Use the Timer control to refresh the content only as often as necessary.
    - You can include more than one Timer control on a Web page if different UpdatePanel controls must be updated at different intervals. Alternatively, a single instance of the Timer control can be the trigger for more than one UpdatePanel control in a Web page.



Timer Control:

- When the **Timer** control is included inside an **UpdatePanel** control, the Timer control automatically works as a trigger for the UpdatePanel control. You can override this behavior by setting the **ChildrenAsTriggers** property of the **UpdatePanel** control to *false*.
- For **Timer** controls inside an **UpdatePanel** control, the **JavaScript timing** component is re-created only when each postback is completed. Therefore, the timed interval does not start until the page returns from the postback. For instance, if the **Interval** property is set to 60,000 milliseconds (60 seconds) but the postback takes 3 seconds to complete, then the next postback will occur 63 seconds after the previous postback.
- The following example shows how to include a Timer control inside an UpdatePanel control.

```
<asp:ScriptManager runat="server" id="ScriptManager1" />
<asp:UpdatePanel runat="server" id="UpdatePanel1"
   UpdateMode="Conditional">
  <contenttemplate>
    <asp:Timer id="Timer1" runat="server"
      Interval="120000"
      OnTick="Timer1_Tick">
    </asp:Timer>
  </contenttemplate>
</asp:UpdatePanel>
```



Timer Control (contd.):
The example in the above slide shows how to include a Timer control inside an UpdatePanel control.

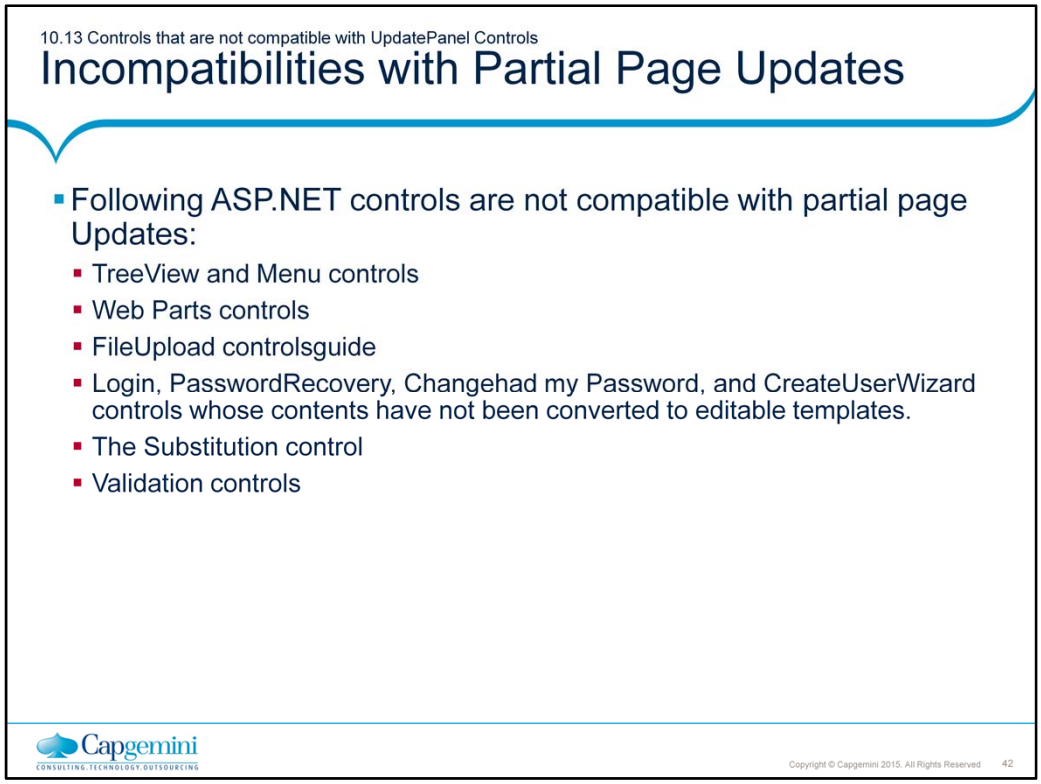
10.12: Timer Control

Demo

- Demo on Using Timer Control

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

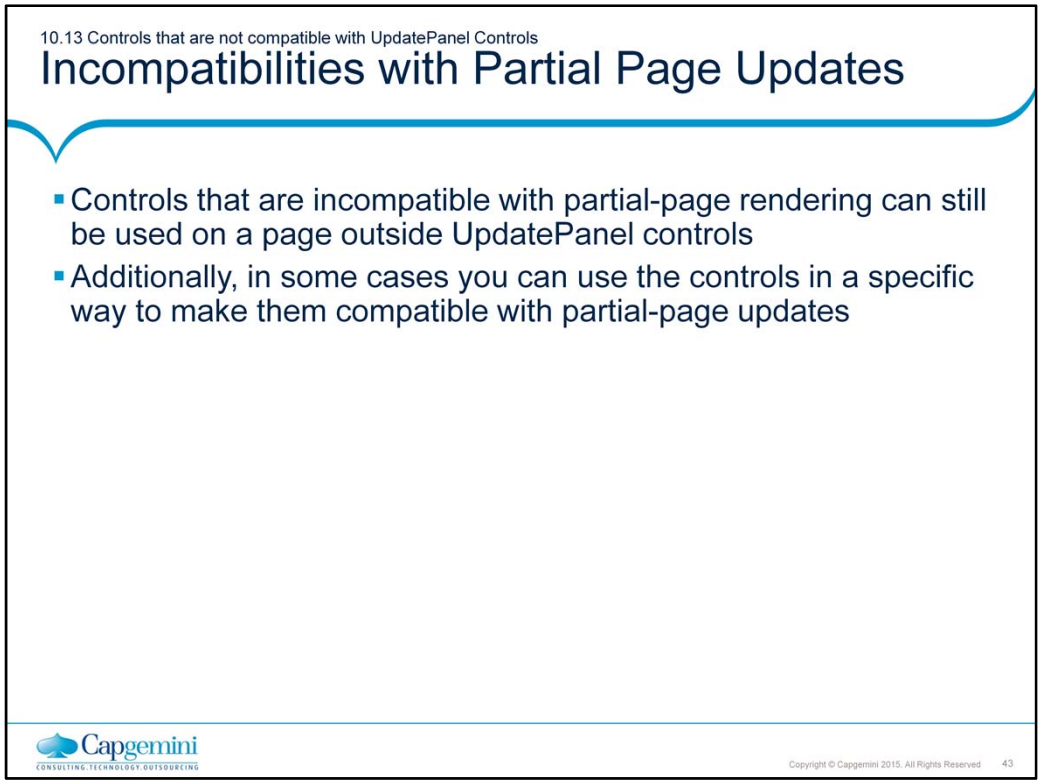


Controls that are not compatible with UpdatePanel Controls:

The following ASP.NET controls are not compatible with partial-page updates, and are therefore not supported inside an UpdatePanel control:

Following are the not compatible controls:

- TreeView and Menu controls
- Web Parts controls
- FileUpload controls when they are used to upload files as part of an asynchronous postback
- GridView and DetailsView controls when their EnableSortingAndPagingCallbacks property is set to true. The default is false.
- Login, PasswordRecovery, ChangePassword, and CreateUserWizard controls whose contents have not been converted to editable templates
- The Substitution control
- Validation controls, which includes the BaseCompareValidator, BaseValidator, CompareValidator, CustomValidator, RangeValidator, RegularExpressionValidator, RequiredFieldValidator, and ValidationSummary control



Controls that Are Not Compatible with UpdatePanel Controls (contd.):

- Controls that are incompatible with partial-page rendering can still be used on a page outside UpdatePanel controls. Additionally, in some cases you can use the controls in a specific way to make them compatible with partial-page updates.
- For example, you can use the Login, ChangePassword, or PasswordRecovery controls inside an UpdatePanel control if you can convert their contents to templates. (If you are using Visual Studio, in Design view you can convert the controls by using smart-tag menu commands such as **Convert to Template** or **Customize Create User Step**.)
- When you convert these controls into editable templates, the validation controls that are used in the control are defined declaratively by using **markup** in the page.
  - To make the **validators** compatible with an UpdatePanel control, set the **EnableClientScript** property of the validators to false. This disables the client script that would ordinarily be used to perform validation in the browser.
  - As a result, during an asynchronous postback, the **validators** perform validation on the server. However, because only the content of the UpdatePanel is refreshed, the validators can provide the kind of immediate feedback that is ordinarily provided by client script.
- To use a **FileUpload control** inside an **UpdatePanel control**, set the **postback control** that submits the file to be a **PostBackTrigger** control for the panel.

Summary

- In this lesson, you have learnt:
  - Three technologies that make AJAX work, namely JavaScript, XMLHttpRequest object, and XML
  - Relation between AJAX and ASP.NET AJAX
  - Most valuable features of ASP.NET AJAX: the UpdatePanel control and related controls – Timer and UpdateProgress
  - The UpdateProgress control implements a wait screen which is a commonly used effect in AJAX applications

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Review Question

- Question 1: Which control manages client script for Microsoft ASP.NET AJAX pages?
- Question 2: Which control you can use to refresh selected parts of the page instead of refreshing the whole page with a postback?
- Question 3: Which property of ScriptManager control is required to set true for partial-page updates?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 45

Review Question

- Question 4: This ___ control provides status information about partial-page updates in UpdatePanel controls
- Question 5: When you use ___ control with an UpdatePanel control, you can enable partial-page updates at a defined interval

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 46