AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'

The **ScriptManager Control**

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

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

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

The **UpdatePanel Control**

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

**Example**

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:
The source file is as follows:

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

  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
      <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial PostBack" />
      <br />
      <br />
      <asp:Label ID="lblpartial" runat="server"></asp:Label>
    </ContentTemplate>
  </asp:UpdatePanel>

  <p>Outside the Update Panel</p>

  <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
  <br />
  <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>

Both the button controls have same code for the event handler:

```csharp
string time = DateTime.Now.ToString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.

```
Partial PostBack
Showing time from panel11:51:31

Outside the Update Panel
Total PostBack
Showing time from outside11:18:10
```

A page can contain multiple update panels with each panel containing other controls like a grid.
and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

**Properties of the UpdatePanel Control**

The following table shows the properties of the update panel control:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildrenAsTriggers</td>
<td>This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.</td>
</tr>
<tr>
<td>ContentTemplate</td>
<td>It is the content template and defines what appears in the update panel when it is rendered.</td>
</tr>
<tr>
<td>ContentTemplateContainer</td>
<td>Retrieves the dynamically created template container object and used for adding child controls programmatically.</td>
</tr>
<tr>
<td>IsInPartialRendering</td>
<td>Indicates whether the panel is being updated as part of the partial post back.</td>
</tr>
<tr>
<td>RenderMode</td>
<td>Shows the render modes. The available modes are Block and Inline.</td>
</tr>
<tr>
<td>UpdateMode</td>
<td>Gets or sets the rendering mode by determining some conditions.</td>
</tr>
<tr>
<td>Triggers</td>
<td>Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.</td>
</tr>
</tbody>
</table>

**Methods of the UpdatePanel Control**

The following table shows the methods of the update panel control:

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateContentTemplateContainer</td>
<td>Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.</td>
</tr>
<tr>
<td>CreateControlCollection</td>
<td>Returns the collection of all controls that are contained in the UpdatePanel control.</td>
</tr>
<tr>
<td>Initialize</td>
<td>Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.</td>
</tr>
<tr>
<td>Update</td>
<td>Causes an update of the content of an UpdatePanel control.</td>
</tr>
</tbody>
</table>

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

<table>
<thead>
<tr>
<th>UpdateMode</th>
<th>ChildrenAsTriggers</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>False</td>
<td>Illegal parameters.</td>
</tr>
<tr>
<td>Always</td>
<td>True</td>
<td>UpdatePanel refreshes if whole page refreshes or a child control on it posts back.</td>
</tr>
</tbody>
</table>
### Conditional False

UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.

### Conditional True

UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

## The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```xml
<asp:UpdateProgress
    ID="UpdateProgress1" runat="server" DynamicLayout="true"
    AssociatedUpdatePanelID="UpdatePanel1" >
    <ProgressTemplate>
        Loading...
    </ProgressTemplate>
</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

## Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssociatedUpdatePanelID</td>
<td>Gets and sets the ID of the update panel with which this control is associated.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Gets or sets the cascading style sheet CSS attributes of the UpdateProgress control.</td>
</tr>
<tr>
<td>DisplayAfter</td>
<td>Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.</td>
</tr>
<tr>
<td>DynamicLayout</td>
<td>Indicates whether the progress template is dynamically rendered.</td>
</tr>
<tr>
<td>ProgressTemplate</td>
<td>Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.</td>
</tr>
</tbody>
</table>

## Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
</table>
GetScriptReferences Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

1. Setting the Triggers property of the UpdatePanel control:

   ```xml
   <Triggers>
   <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
   </Triggers>
   ```

2. Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

   ```xml
   <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">
   <ContentsTemplate>
   <asp:Timer ID="Timer1" runat="server" Interval="1000"></asp:Timer>
   <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" />
   </ContentsTemplate>
   </asp:UpdatePanel>
   ```