ASP.NET AJAX, previously called "Atlas", is a Microsoft implementation of an AJAX based framework, created for ASP.NET (although it can be used on other platforms as well). AJAX stands for Asynchronous JavaScript and XML, which, very simply put, is a way of transferring data between the server and client without sending the entire page, and thereby creating a complete postback. This allows for a richer experience for the user, since loading dynamic content can be done in the background, without refreshing and redrawing the entire page. If you have ever used Gmail or Outlook Web Access, you have used an Ajax enabled webapplication, and especially Google have made Ajax very popular.   
  
While it's perfectly possible to use Ajax without Microsoft ASP.NET AJAX, a lot of things are way easier, since Microsoft has wrapped some of most tedious parts of Ajax into their implementation. For instance, the 3 most popular browsers requires different ways of using Ajax, and have different JavaScript implementations. ASP.NET AJAX simplifies this a lot and allows you to write the same code to target all 3 major browsers.

[ASP.NET tutorial](http://asp.net-tutorials.com/). It will get you started. Using ASP.NET AJAX gets a lot easier if you have a proper IDE, and we recommend Visual Studio Express 2012 for Web, as described in our ASP.NET tutorial.   
  
After you've installed the latest version, start up "Visual Studio Express 2012 for Web", and use the template "ASP.NET website" when you select "New Web Site..." from the File menu.   
  
Read on in the following chapters, where we will get you started with ASP.NET AJAX.

As usual, we will use the good old "Hello, world!" as our very first example. We will begin with the code, and then we'll do a bit of explanation afterwards. If you haven't already done so, you should create a new ASP.NET website project in Visual Studio . The IDE will create a Default.aspx and Default.aspx.cs file for you, which will look just like any other ASP.NET enabled page. Let's add some AJAX to it:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>

<!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>Hello, world!</title>

</head>

<body>

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

<asp:ScriptManager ID="MainScriptManager" runat="server" />

<asp:UpdatePanel ID="pnlHelloWorld" runat="server">

<ContentTemplate>

<asp:Label runat="server" ID="lblHelloWorld" Text="Click the button!" />

<br /><br />

<asp:Button runat="server" ID="btnHelloWorld" OnClick="btnHelloWorld\_Click" Text="Update label!" />

</ContentTemplate>

</asp:UpdatePanel>

</form>

</body>

</html>

In the CodeBehind, there's nothing new except for this event which you should add:

protected void btnHelloWorld\_Click(object sender, EventArgs e)

{

lblHelloWorld.Text = "Hello, world - this is a fresh message from ASP.NET AJAX! The time right now is: " + DateTime.Now.ToLongTimeString();

}

In the markup part, we use two new things, when compared to regular ASP.NET: The ScriptManager control and the UpdatePanel control. The ScriptManager makes sure that the required ASP.NET AJAX files are included and that AJAX support is added, and has to be included on every page where you wish to use AJAX functionality.   
  
After the manager, we have one of the most used controls when working with AJAX, the UpdatePanel. This control allows you to wrap markup which you would like to allow to be partially updated, that is, updated without causing a real postback to the server. More about the UpdatePanel in a coming chapter. Besides those two controls, everything else is standard controls, with no modifications that would indicate alternate behavior.

Try running the example site, and click the button. The label will be updated with our usual Hello world text, and the current time. Try repeatedly clicking the button, and you will see the label get the current timestamp each time. Notice the wonderful absence of a blinking window and a running status bar - everything is done without updating anything but the label! We've just created our first AJAX enabled page. If you wish to see how this page would work without AJAX, try setting the "enablepartialrendering" of the ScriptManager to false like this:

<asp:ScriptManager ID="MainScriptManager" runat="server" enablepartialrendering="false" />

This will disallow the use of partial rendering on the page, and show you how it would work without AJAX.   
  
In the following chapters we will look into the various AJAX controls and how to use them.

**UpdatePanel control**

The UpdatePanel control is probably the most important control in the ASP.NET AJAX package. It will AJAX'ify controls contained within it, allowing partial rendering of the area. We already used it in the Hello world example, and in this chapter, we will go in depth with more aspects of the control.   
  
The <asp:UpdatePanel> tag has two childtags - the **ContentTemplate** and the **Triggers** tags. The ContentTemplate tag is required, since it holds the content of the panel. The content can be anything that you would normally put on your page, from literal text to web controls. The Triggers tag allows you to define certain triggers which will make the panel update it's content. The following example will show the use of both childtags.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>UpdatePanel</title>

</head>

<body>

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

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

<asp:UpdatePanel runat="server" id="UpdatePanel" updatemode="Conditional">

<Triggers>

<asp:AsyncPostBackTrigger controlid="UpdateButton2" eventname="Click" />

</Triggers>

<ContentTemplate>

<asp:Label runat="server" id="DateTimeLabel1" />

<asp:Button runat="server" id="UpdateButton1" onclick="UpdateButton\_Click" text="Update" />

</ContentTemplate>

</asp:UpdatePanel>

<asp:UpdatePanel runat="server" id="UpdatePanel1" updatemode="Conditional">

<ContentTemplate>

<asp:Label runat="server" id="DateTimeLabel2" />

<asp:Button runat="server" id="UpdateButton2" onclick="UpdateButton\_Click" text="Update" />

</ContentTemplate>

</asp:UpdatePanel>

</form>

</body>

</html>

Here is the CodeBehind. Just add the following method to the file:

protected void UpdateButton\_Click(object sender, EventArgs e)

{

DateTimeLabel1.Text = DateTime.Now.ToString();

DateTimeLabel2.Text = DateTime.Now.ToString();

}

So, what's this example all about? Try running it, and click the two buttons. You will notice that then first button updates only the first datestamp, while the second button updates both. Why? We have set the Panels to update conditionally, which means that their content is only updated if something insides them causes a postback, or if one of the defined triggers are fired.

As you can see, the first UpdatePanel carries a trigger which references the second button. This will ensure that the first panel is updated even when a control on a different UpdatePanel is used.   
  
The AsyncPostBackTrigger tag is pretty simple - it only takes two attributes, the controlid, a reference to the control which can trigger it, and the eventname, which tells which eventtype can cause the trigger to fire. If you wish for the content of a UpdatePanel to be updated no matter what, you may change the updatemode property to Always.   
  
In general, you should only have UpdatePanels areound areas where you wish to do partial updates. Don't wrap your entire page within an UpdatePanel, and don't be afraid to use several panels, since this will give you more control of which areas update and when they do it