**Using Custom BasePage Class in ASP.Net**

Every Code Behind class of ASP.Net pages will inherit System.Web.UI.Page Class by default.

Refer the below code.

public partial class Register : System.Web.UI.Page

{

}

 In this inheritance hierarchy we can include a custom BasePage class that inherits System.UI.Page class and in turn make the codebehind class to inherit the BasePage Class.

When the value of the AutoEventWireup attribute is false

If you want to manually hook up events to an event handler, set the value of the **AutoEventWireup** attribute to **false**. The following sample shows the code that you can use to handle the **Load** event of the **Page** object in an ASP.NET Web Form:

1. Start Microsoft Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, under **Project Types**, click **Visual C# Projects**. Under **Templates**, click **ASP.NET Web Application**.
4. In the **Location** box, type the project name as **http://*ServerName*/MyWebApp**.   
     
   **Note** Replace *ServerName* with the name of a server. MyWebApp is the name of a sample ASP.NET Web Application.
5. In Solution Explorer, right-click the **WebForm1.aspx** file, click **Rename**, and then type **EventWireUpFalse.aspx**.
6. Replace the existing code in the EventWireUpFalse.aspx file with the following code:
7. <%@ Page Language="C#" AutoEventWireup="false" Inherits="MyWebApp.EventWireUpFalse" %>
8. <HTML>
9. <HEAD>
10. <title>Visual C# .NET WIRE-UP FALSE</title>
11. </HEAD>
12. <BODY>
13. <p><% Response.Write(message); %></p>
14. </BODY>

</HTML>

1. Replace the existing code in the EventWireUpFalse.aspx.cs file with the following code:
2. using System;
3. namespace MyWebApp
4. {
5. public class EventWireUpFalse : System.Web.UI.Page
6. { public string message;
7. private void Page\_Load(object sender, System.EventArgs e)
8. {
9. message="The Page\_Load Event Fired with AutoEventWireup False";
10. }
11. // Visual C# .NET requires that you override the OnInit function,
12. // adding a new delegate for the Page\_Load event.
13. override protected void OnInit(EventArgs e)
14. {
15. this.Load += new System.EventHandler(this.Page\_Load);
16. }
17. }

}

1. On the **Debug** menu, click **Start** to build and to run the ASP.NET Web application.  
     
   In this example, you receive a message when the ASP.NET page framework raises the **Page\_Load** event handler. If the value of the **AutoEventWireup** attribute is set to **false**, you must override the **OnInit** function, and then you must add a new delegate for the **Page\_Load** event handler.

When the value of the AutoEventWireup attribute is true

When you set the value of the **AutoEventWireup** attribute to **false**, you must manually hook up events to event handlers. When you set the value of the **AutoEventWireup** attribute to **true**, the ASP.NET page framework can automatically raise events. The following sample describes how to code a **Page\_Load** event handler in an ASP.NET Web Form when the value of the**AutoEventWireup** attribute is **true**.

1. Start Microsoft Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, under **Project Types**, click **Visual C# Projects**. Under **Templates**, click **ASP.NET Web Application**.
4. In the **Location** box, type the project name as **http://*ServerName*/MyWebApp**.   
     
   **Note** Replace *ServerName* with the name of a server. MyWebApp is the name of a sample ASP.NET Web Application.
5. In Solution Explorer, right-click the **WebForm1.aspx** file, click **Rename**, and then type **EventWireUpTrue.aspx**.
6. Replace the existing code in the EventWireUpTrue.aspx file with the following code:
7. <%@ Page Language="C#" AutoEventWireup="true" Inherits="MyWebApp.EventWireUpTrue" %>
8. <HTML>
9. <HEAD>
10. <title>Visual C# .NET WIRE-UP TRUE</title>
11. </HEAD>
12. <BODY>
13. <p><% Response.Write(message); %></p>
14. </BODY>

</HTML>

1. Replace the existing code in the EventWireUpTrue.aspx.cs file with the following code:
2. using System;
3. namespace MyWebApp
4. {
5. public class EventWireUpTrue : System.Web.UI.Page
6. { public string message;
7. private void Page\_Load(object sender, System.EventArgs e)
8. {
9. message="The Page\_Load Event fired with AutoEventWireup True";
10. }
11. }

}

1. On the **Debug** menu, click **Start** to build and run the project.  
     
   In this example, you receive a message when the ASP.NET page framework raises the **Page\_Load** event handler. If the value of the **AutoEventWireup** attribute is **true**, you do not have to override the **OnInit** function, and you do not have to add a new delegate for the **Page\_Load** event handler.

**AutoEventWireup attribute in ASP.NET**  
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"Inherits="\_Default" %>  
  
The ASP.NET page framework supports an automatic way to associate page events and methods. If the AutoEventWireup attribute of the Page directive is set to true, the page framework calls page events automatically, specifically the Page\_Init and Page\_Load methods. In that case, no explicit Handles clause or delegate is needed.

* AutoEventWireup is an attribute in Page directive.
* AutoEventWireup is a Boolean attribute that indicates whether the ASP.NET pages events are auto-wired.
* AutoEventWireup will have a value true or false. By default it is true.

There is no event or method associated with Page\_Load. Those events whose inline event is not there but that should be executed, for that purposed AutoEventWireup="true".  
  
**Disadvantages of AutoEventWireup attribute**

* AutoEventWireup uses fixed naming convention for the events. Page events handlers have specific predictable names. This limits your flexibility in how you name event handlers.
* If you do set AutoEventWireup to true, Visual Studio will generate code to bind the events and the page framework will automatically call events based on their names. This can result in the same event code being called twice when the page runs. As a consequence, you should always leave AutoEventWireup set to false when working in Visual Studio.
* Another disadvantage is that performance is adversely affected, because ASP.NET searches for methods at run-time. For a Web site with high traffic volumes, the impact on performance could be significant.

AutoEventWireup="true" target is for page events only. In case of AutoEventWireup method are not case sensitive. (Page\_Load or page\_load both will work).  
  
If AutoEventWireup="false" but still you want to executed Page event (Page\_Load). In this you have to explicitly code for it.  
  
<form id="form1" runat="server" onload="Page\_Load">

**Update Panel**

The content of an [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control is updated in the following circumstances:

* If the UpdateMode property is set to [Always](http://msdn.microsoft.com/en-us/library/bb155262(v=vs.110).aspx), the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) 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](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) controls and postbacks from controls that are not inside [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) controls.
* If the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control is nested inside another [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control and the parent update panel is updated.
* If the UpdateMode property is set to [Conditional](http://msdn.microsoft.com/en-us/library/bb155262(v=vs.110).aspx), and one of the following conditions occurs:
  + You call the [Update](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.update(v=vs.110).aspx) method of the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control explicitly.
  + The postback is caused by a control that is defined as a trigger by using the [Triggers](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.triggers(v=vs.110).aspx) property of the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control. In this scenario, the control explicitly triggers an update of the panel content. The control can be either inside or outside the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control that defines the trigger.
  + The [ChildrenAsTriggers](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.childrenastriggers(v=vs.110).aspx) property is set to **true** and a child control of the [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control causes a postback. A child control of a nested [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx)control does not cause an update to the outer [UpdatePanel](http://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel(v=vs.110).aspx) control unless it is explicitly defined as a trigger.

**1. AsyncPostBackTrigger**

it is the one which enforces Asynchonous post back of the Page.., i.e. the AJAX way. The data will be transacted without full post back. When you are using functionalities like login, you may use this.

Ex. You are having two dropDowns Viz., Countries and States. the states should be loaded when a country is selected and it should be changed on Countries change.

You may use AsyncPostBackTrigger in this scinario., which will populate the states ddl without full post back.

**2. PostBackTrigger**

It is the one which does not follow the AJAX functioalities., but the full post back as usually(as Without using UpdatePanel). Situtions are there where you would not like to enforce Partial Post back (as explained in Point 1. above). Like you are having FileUpload Control withing the UpdatePanel and when you do it by AsyncPostBack, you will not get any values to the server. It requires Full PostBack. in such a case you should use this trigger.

<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>