**ASP.NET page life cycle Events**

<http://www.c-sharpcorner.com/uploadfile/61b832/Asp-Net-page-life-cycle-events/>

**(1) PreInit**

The entry point of the page life cycle is the pre-initialization phase called "PreInit". This is the only event where programmatic access to master pages and themes is allowed.  You can dynamically set the values of master pages and themes in this event.  You can also dynamically create controls in this event.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

public partial class \_Default : System.Web.UI.Page

{

    protected void Page\_PreInit(object sender, EventArgs e)

    {

        //  Use this event for the following:

      //  Check the IsPostBack property to determine whether this is the first time the page is being processed.

        //  Create or re-create dynamic controls.

        //  Set a master page dynamically.

        //  Set the Theme property dynamically.

    }

**(2)Init**

This event fires after each control has been initialized, each control's UniqueID is set and any skin settings have been applied. You can use this event to change initialization values for controls. The "Init" event is fired first for the most bottom control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Page\_Init(object sender, EventArgs e)

{

    // Raised after all controls have been initialized and any skin settings have been applied.   
    // Use this event to read or initialize control properties.

}

**(3)InitComplete**

Raised once all initializations of the page and its controls have been completed. Till now the viewstate values are not yet loaded, hence you can use this event to make changes to view state that you want to make sure are persisted after the next postback

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Page\_InitComplete(object sender, EventArgs e)

{

       // Raised by the  Page object. Use this event for processing tasks that require all initialization be complete.

}

**(4)PreLoad**

Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance

Loads ViewState : ViewState data are loaded to controls

Note : The page viewstate is managed by ASP.NET and is used to persist information over a page roundtrip to the server. Viewstate information is saved as a string of name/value pairs and contains information such as control text or value. The viewstate is held in the value property of a hidden <input> control that is passed from page request to page request.

Loads Postback data : postback data are now handed to the page controls

Note : During this phase of the page creation, form data that was posted to the server (termed postback data in ASP.NET) is processed against each control that requires it. Hence, the page fires the LoadPostData event and parses through the page to find each control and updates the control state with the correct postback data. ASP.NET updates the correct control by matching the control's unique ID with the name/value pair in the NameValueCollection. This is one reason that ASP.NET requires unique IDs for each control on any given page.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected override void OnPreLoad(EventArgs e)

{

        // Use this event if you need to perform processing on your page or control before the  Load event.

        // Before the Page instance raises this event, it loads view state for itself and all controls, and   
        // then processes any postback data included with the Request instance.

}

**(5)Load**

The important thing to note about this event is the fact that by now, the page has been restored to its previous state in case of postbacks. Code inside the page load event typically checks for PostBack and then sets control properties appropriately. This method is typically used for most code, since this is the first place in the page lifecycle that all values are restored. Most code checks the value of IsPostBack to avoid unnecessarily resetting state. You may also wish to call Validate and check the value of IsValid in this method. You can also create dynamic controls in this method.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Page\_Load(object sender, EventArgs e)

{

        // The  Page calls the  OnLoad event method on the  Page, then recursively does the same for each child  
        // control, which does the same for each of its child controls until the page and all controls are loaded.

        // Use the OnLoad event method to set properties in controls and establish database connections.

}

**(6)Control (PostBack) event(s)**

ASP.NET now calls any events on the page or its controls that caused the PostBack to occur. This might be a button's click event or a dropdown's selectedindexchange event, for example.

These are the events, the code for which is written in your code-behind class(.cs file).

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Button1\_Click(object sender, EventArgs e)

{

        // This is just an example of control event.. Here it is button click event that caused the postback

}

**(7)LoadComplete**

This event signals the end of Load.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Page\_LoadComplete(object sender, EventArgs e)

{

        // Use this event for tasks that require that all other controls on the page be loaded.

}

**(8)PreRender**

Allows final changes to the page or its control. This event takes place after all regular PostBack events have taken place. This event takes place before saving ViewState, so any changes made here are saved.

For example : After this event, you cannot change any property of a button or change any viewstate value. Because, after this event, SaveStateComplete and Render events are called.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected override void OnPreRender(EventArgs e)

{

        // Each data bound control whose DataSourceID property is set calls its DataBind method.

        // The PreRender event occurs for each control on the page. Use the event to make final   
        // changes to the contents of the page or its controls.

}

**(9)SaveStateComplete**

Prior to this event the view state for the page and its controls is set.  Any changes to the page's controls at this point or beyond are ignored.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected override void OnSaveStateComplete(EventArgs e)

{

        // Before this event occurs,  ViewState has been saved for the page and for all controls.   
        // Any changes to the page or controls at this point will be ignored.

        // Use this event perform tasks that require view state to be saved, but that do not make any changes to controls.

}

**(10)Render**

This is a method of the page object and its controls (and not an event). At this point, ASP.NET calls this method on each of the page's controls to get its output. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

Note: Right click on the web page displayed at client's browser and view the Page's Source. You will not find any aspx server control in the code. Because all aspx controls are converted to their respective HTML representation. Browser is capable of displaying HTML and client side scripts.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

// Render stage goes here. This is not an event

**(11)UnLoad**

This event is used for cleanup code. After the page's HTML is rendered, the objects are disposed of. During this event, you should destroy any objects or references you have created in building the page. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.

Cleanup can be performed on-

(a)Instances of classes i.e. objects

(b)Closing opened files

(c)Closing database connections.

EXAMPLE : Override the event as given below in your code-behind cs file of your aspx page

protected void Page\_UnLoad(object sender, EventArgs e)

{

        // This event occurs for each control and then for the page. In controls, use this event to do final cleanup   
        // for specific controls, such as closing control-specific database connections.

        // During the unload stage, the page and its controls have been rendered, so you cannot make further   
        // changes to the response stream.

//If you attempt to call a method such as the Response.Write method, the page will throw an exception.

    }

}