|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **Introduction**  It is very important to **understand** **ASP.NET** **page** life cycle for many reasons, mainly for **understand**ing where we should place particular methods and when we should set **page** related properties. Additionally if we are developing custom server controls, then clear **understand**ing of **page** life cycle is very useful to correct initialization of control, setting properties with view-state data, and control’s code (Control events are subject to **ASP.NET** **page** only).  **Page** life cycle depends on whether it is requested for the first time or it is after postback (**page** request of itself) and finalize to the web server. When a web **page** is requested to the web server, it goes through a series of sequence of steps/events (like initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering) before it is returned back to the web browser.  When we use and manipulate **page** life cycle events correctly, then it becomes a handy and powerful tool for developing web applications.  **Background**  **IIS**: It is the default web server with Microsoft .NET. Internet Information Server helps to deploy web sites/web application. IIS web server receives requests for a web resource (file), it checks the extension of the file (e.g. .aspx, .ascx, .ashx and .asmx) and then it determines which ISAPI extension should handle this request and passes this request to proper ISAPI extension.  **ASPNET\_ISAPI.DLL**: IIS loads this DLL and sends the **page** request to this DLL. This DLL loads the HTTPRuntimefor further processing.  **ASPNET\_WP.EXE**: It contains an Application Pool. Each Application Pool can contain any number of Applications. Application Pool is also called as AppDomain. When a web **page** is requested, IIS looks for the application pool under which the current application is running and forwards the request to the respective worker process.  **Explanation**   * **PreInit** * **Init** * **InitComplete** * **PreLoad** * **Load** * **Control Events** * **LoadComplete** * **PreRender** * **SaveViewState** * **Render** * **Unload**   General **Page** Life Cycle Stages each and every time the browser sends the request, **page** instance is created. HTTP runtime invokes ProcessRequest and starts **page** execution.  **Table Showing Stage and Corresponding Events**   | **Stage** | **Events/Method** | | --- | --- | | Initialization of the **page** | **Page**\_Init | | Loading of the View State | LoadViewState | | Processing of the Postback data | LoadPostData | | Loading of **Page** | **Page**\_Load | | Notification of PostBack | RaisePostDataChangedEvent | | Handling of PostBack Event | RaisePostBackEvent | | Pre Rendering of **Page** | **Page**\_PreRender | | Saving of view state | SaveViewState | | Rendering of **Page** | **Page**\_Render | | Unloading of the **Page** | **Page**\_UnLoad |     **Setting Up Of the Page**  **ASP.NET** determines that the **page** request by a user requires to be parsing and compiling or to render cached copy of the **page** to be sent. Thus it comes very much before the beginning of the **page** life cycle. After this, it is also checked that request is a normal request, postback, cross-**page** postback or callback. The **page** constructor creates a tree of controls as soon as the HTTP runtime instantiates the **page** class to perform the request.  **Events**  **PreInit**  This event is the beginning of the **page** life cycle.   * Possible to change or set Master **page**, themes * Creates or re-creates dynamic controls * Reads or sets Profile property values   **Init**  First, the Init event for the **Page** object occurs, then Init event occurs for each control on the **Page**. Viewstate information is not available at this stage.   * Controls have been initialized * Theme skins applied if any * Initialize control properties   **InitComplete**  This event is used for processing tasks that require all initialization to be complete.  **PreLoad**  This event is used before performing any processing that should occur before Load. 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.  **Load**  Set properties in controls and establish database connections.  **Control Events**  These are control specific events such as – Button Click, DropDownIndexChanged, etc.  **Load Complete**  This event is used for performing those tasks which require load has been completed.  **PreRender**  In this event, **Page** ensures that all child controls are created. **Page** calls EnsureChildControls for all controls, including itself. Every control whose datasource/databind property is set calls for its databind method.  **SaveStateComplete**  This event occurs after viewstate is encoded and saved for the **page** and for all controls.  **Render**  Every **ASP.NET** control has render method and the **page** instance calls this method to output the control’s markup, after this event any changes to the **page** or controls are ignored.  **Unload**  Unload event is used to do cleanup tasks like closing the database connections, closing of the open files, completing logging or other requested tasks. Unload events occurs for each control on **page** control tree first and after that**page**. |
| **2** | **HTML Server Control:** Traditional HTML tags – System.Web.UI.HtmlControls   * HTML Controls are by default treated as text in ASP.NET. To make these elements programmable add runat=”server” attribute to the HTML element. * If the requirement is to program the control on client side in javascript and has no code to execute on the server then only we should go for HTML controls otherwise always WebServer controls must used because they are very dynamic, powerful and also has enhanced functionality.   **Web server Controls:** Web server controls are special ASP.NET tags understand by the ASP.NET server.  All these controls are under the namespace – System.Web.UI.WebControls  **Label:** Renders span tag to the client.  **Properties:** Text  **Literal Control:** Renders just text without any tag and henced cannot be formatted or programmed in browser.  **Properties:** Text  **Textbox:** Can render either <input type=”text”> or <input type=”password”> or <textarea> tags.  **Properties:** Text, TextMode, ReadOnly, Maxlength, AutoPostBack.  **Event:** TextChaange  **Button:** render as <input type=”submit”> tag.  **Properties:** Text, OnClick, OnClientClick.  **Event:** Click  **e.g.** <asp:Button OnClientClick=”return confirm(“Are you sure”)”…/>  is rendered as <input type=”submit” OnClick=”return confirm(“Are you sure”)…/>  **Note:** In javascript OnClick of Submit Button retrns falsem the form is not submitted to server.  **LinkButton:** render as <A href=”\_\_doPostBack(…)”…>. Used for pointing the form to server.  **Properties:** Text, OnClientClick  **Event:** Click  **Hyperlink:** render as <A href=”Url”…> used for linking to another page and not posting the form.  **Properties:** Text, ImageUrl, NavigateUrl, Target.  **ImageButton:** render as <input type=”image”…>  **Properties:** ImageUrl, OnClick  **CheckBox:** render as <input type=”Checkbox”…/>  **Properties:** Text, TextAllign ,Checked, AutoPostBack  **Event:** CheckedChanged  **RadioButton:** render as <input type=”radio”>. It inherited from CheckBox class  **Properties:** Text, Checked, AutoPostBack, GroupName (must be same for grouping the radio buttons).  **Event:** CheckedChanged  **ListControl:** It’s common parent for DropdownList, ListBox, CheckedBoxList, RadioButtonList, BulletedList  **Properties:** Items, SelectedIndex, Selecteditem, SelectedValue, Text, AutoPostBack  **Event:** SelectedIndexChanged  **Methods:** ClearSelection  **ListItem:** renders based on container tag in which it is used.  **Properties:** Text, Value, Selected  **DropDownList:** render as <select > tag.  **ListBox:** render as <select size=”4”….> tag  **Properties:** Rows (maps to size attribute), SelectionMode(Single/Multiple)  **CheckedboxList:** renders <input type=”checkbox”….> for every ListItem.  **Properties:** RepeatColumn, RepeatDirection, RepeatLayout.  **RadioButtonList:** renders as < input type=”radio”…. > for every listItem.  **Properties:** RepeatColumn, RepeatDirection, RepeatLayout. |
| **3** | **ASP.NET 4 includes six validation controls**  **RequiredFieldValidater :** Enable you to require a user to enter a value in a for form field.  **Properties:** Text, ErrorMessage, ControlToValidate, Display, SetFocusOnError,ValidationGroup  Example:  <asp:Label ID="lblUsername" runat="server" Text="Username"></asp:Label>  <asp:TextBox ID="txtusername" runat="server"></asp:TextBox>  <asp:RequiredFieldValidator ID="rfvUsername" runat="server" ErrorMessage="Username Required" Text="Required" ControlToValidate="txtUsername"></asp:RequiredFieldValidator>  <br />  <asp:Label ID="lblPassword" runat="server" Text="Password"></asp:Label>  <asp:TextBox ID="txtpassword" runat="server" TextMode="Password"></asp:TextBox>  <asp:RequiredFieldValidator ID="rfvPassword" runat="server" ErrorMessage="Password Required" Text="Required" ControlToValidate="txtPassword"></asp:RequiredFieldValidator>  <br />  <asp:Button ID="btnLogin" runat="server" Text="Login" />  **RangeValidater:** Enables you to check whether a value falls between a certain minimum and maximum value.  **Properties:** ControlToValidate, Text, MinimumValue, MaximumValue, Type  Example:  <asp:Label ID="lblAge" runat="server" Text="Age"></asp:Label>  <asp:TextBox ID="txtAge" runat="server"></asp:TextBox>  <asp:RangeValidator ID="rvAge" runat="server" ErrorMessage="Age must be between 1 to 10" MaximumValue="10" MinimumValue="1" Type="Integer" ControlToValidate="txtAge"></asp:RangeValidator>  <asp:Button ID="btn" runat="server" Text="Button" />  **CompareValidater:** Enable you to compare a value against another value or perform a data type check.  **Properties:** ControlToValidate, Text, Type, Operator, ValueToCompare, ControlToCompare  Example:  <asp:Label ID="lblAge" runat="server" Text="Age"></asp:Label>  <asp:TextBox ID="txtAge" runat="server"></asp:TextBox>  <asp:RangeValidator ID="rvAge" runat="server" ErrorMessage="Age must be between 1 to 10" MaximumValue="10" MinimumValue="1" Type="Integer" ControlToValidate="txtAge"></asp:RangeValidator>  <br />  <asp:Label ID="lblConfirmAge" runat="server" Text="Confirm Age"></asp:Label>  <asp:TextBox ID="txtConfirmAge" runat="server"></asp:TextBox>  <asp:CompareValidator ID="cvConfirmAge" runat="server" ErrorMessage="Age Mismatch" ControlToValidate="txtConfirmAge" ControlToCompare="txtAge" Operator="Equal"></asp:CompareValidator>  <asp:Button ID="btn" runat="server" Text="Button" />  </div>  </form>  **RegularExpressionValidater:** Enable you to compare value against a regular expression.  **Properties:**ControlToValidate, Text, ValidationExpression  Example  <asp:Label ID="lblPassword" runat="server" Text="Password"></asp:Label>  <asp:TextBox ID="txtPassword" runat="server" TextMode="Password"></asp:TextBox>  <asp:RegularExpressionValidator ID="revPasswordLength" runat="server" ErrorMessage="weak password" ControlToValidate="txtPassword" ValidationExpression="\w\w\w\w\w\w\w\*"></asp:RegularExpressionValidator>  <br />  <asp:Button ID="btnPassword" runat="server" Text="submit" />  **CustomValidater:**Enables you to perform custom validation.  **Properties:**ControlToValidate, Text, ClientValidationFunction, ServerValidation  Example  <script runat="server">  void comment\_validaton(object sender, ServerValidateEventArgs e)  {  if (e.Value.Length > 10)  {  e.IsValid = false;  }  else  {  e.IsValid = true;  }  }  </script>  <asp:Label ID="lblComment" runat="server" Text="Comment"></asp:Label>  <br />  <asp:TextBox ID="txtComment" runat="server" TextMode="MultiLine" Rows="3" Columns="30"></asp:TextBox>  <asp:CustomValidator ID="csvComment" runat="server" ErrorMessage="Comment is too long" ControlToValidate="txtComment" OnServerValidate="comment\_validaton"></asp:CustomValidator>  <br />  <asp:Button ID="btnComment" runat="server" Text="Comment" />  **ValidationSummary:** Enables you to display a summary of all validation error in a page  **Propertie s:** DisplayMode, HeaderText, ShowMessageBox, ShowSummary  Example:  <asp:ValidationSummary ID="vsLogin" runat="server" ShowSummary="false" ShowMessageBox="true" DisplayMode="BulletList" HeaderText="Error"/>  <asp:Label ID="lblUsername" runat="server" Text="Username"></asp:Label>  <asp:TextBox ID="txtusername" runat="server"></asp:TextBox>  <asp:RequiredFieldValidator ID="rfvUsername" runat="server" ErrorMessage="Username Required" Text="Required" ControlToValidate="txtUsername"></asp:RequiredFieldValidator>  <br />  <asp:Label ID="lblPassword" runat="server" Text="Password"></asp:Label>  <asp:TextBox ID="txtpassword" runat="server" TextMode="Password"></asp:TextBox>  <asp:RequiredFieldValidator ID="rfvPassword" runat="server" ErrorMessage="Password Required" Text="Required" ControlToValidate="txtPassword"></asp:RequiredFieldValidator>  <br />  <asp:Button ID="btnLogin" runat="server" Text="Login" /> |
| **4** | **What is an Event?**   * Event is an action or occurrence like mouse click, key press, mouse movements, or any system generated notification. The processes communicate through events. For example, Interrupts are system generated events. When events occur the application should be able to respond to it. * In ASP.Net an event is raised on the client, and handled in the server. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server. * The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler, and if it has, the event handler is executed.   **Event Arguments:**   * ASP.Net event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is called the event argument. * The general syntax of an event is:   + private void EventName (object sender, EventArgs e);   **Application and Session Events:**  The most important application events are:   * Application\_Start . it is raised when the application/website is started * Application\_End . it is raised when the application/website is stopped   Similarly, the most used Session events are:   * Session\_Start . it is raised when a user first requests a page from the application * Session\_End . it is raised when the session ends   **Page and Control Events:**  Common page and control events are:   * DataBinding . raised when a control bind to a data source * Disposed . when the page or the control is released * Error . it is an page event, occurs when an unhandled exception is thrown * Init . raised when the page or the control is initialized * Load . raised when the page or a control is loaded * PreRender . raised when the page or the control is to be rendered * Unload . raised when the page or control is unloaded from memory   **Event Handling Using Controls:**  All ASP.Net controls are implemented as classes, and they have events which are fired when user performs certain action on them. For example, when a user clicks a button the 'Click' event is generated. For handling these events there are in-built attributes and event handlers. To respond to an event, the event handler is coded.  By default Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.  **The asp tag for a button control:**  **<asp:Button ID="btnCancel" runat="server" Text="Cancel" Onclick="btnCancel\_Click" />**  **The event handler for the Click event:**  **protected void btnCancel\_Click(object sender, EventArgs e)**  **The common control events are:**   |  |  |  | | --- | --- | --- | | **Event** | **Attribute** | **Controls** | | Click | OnClick | Button, image button, link button, image map | | Command | OnCommand | Button, image button, link button | | TextChanged | OnTextChanged | Text box | | SelectedIndexChanged | OnSelectedIndexChanged | Drop-down list, list box, radio button list, check box list. | | CheckedChanged | OnCheckedChanged | Check box, radio button |   Some events cause the form to be posted back to the server immediately; these are called the postback events. For example, the click events like, Button.Click. Some events are not posted back to the server immediately, these are called non-postback events.  For example, the change events or selection events, such as, TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.  **Default Events:**  The default event for the Page object is the Load event. Similarly every control has a default event. For example, default event for the button control is the Click event.  The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:   |  |  | | --- | --- | | **Control** | **Default Event** | | AdRotator | AdCreated | | BulletedList | Click | | Button | Click | | Calender | SelectionChanged | | CheckBox | CheckedChanged | | CheckBoxList | SelectedIndexChanged | | DataGrid | SelectedIndexChanged | | DataList | SelectedIndexChanged | | DropDownList | SelectedIndexChanged | | HyperLink | Click | | ImageButton | Click | | ImageMap | Click | | LinkButton | Click | | ListBox | SelectedIndexChanged | | Menu | MenuItemClick | | RadioButton | CheckedChanged | | RadioButtonList | SelectedIndexChanged |   **Example:**  This example has a simple page with a label control and a button control on it. As the page events like, Page\_Load, Page\_Init, Page\_PreRender etc. takes place, it sends a message, which is displayed by the label control. When the button is clicked, the Button\_Click event is raised and that also sends a message to be displayed on the label.  Create a new website and drag a label control and a button control on it from the control tool box. Using the properties window, set the IDs of the controls as .lblmessage. and .btnclick. respectively. Set the Text property of the Button control as 'Click'.  The markup file (.aspx):   |  | | --- | | <%@ Page Language="C#" AutoEventWireup="true"  CodeBehind="Default.aspx.cs"  Inherits="eventdemo.\_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>Untitled Page</title>  </head>  <body>  <form id="form1" runat="server">  <div>  <asp:Label ID="lblmessage" runat="server" >  </asp:Label>  <br />  <br />  <br />  <asp:Button ID="btnclick" runat="server" Text="Click"  onclick="btnclick\_Click" />  </div>  </form>  </body>  </html> |   Double click on the design view to move to the code behind file. The Page\_Load event is automatically created without any code in it. Write down the following self-explanatory code lines:   |  | | --- | | using System;  using System.Collections;  using System.Configuration;  using System.Data;  using System.Linq;  using System.Web;  using System.Web.Security;  using System.Web.UI;  using System.Web.UI.HtmlControls;  using System.Web.UI.WebControls;  using System.Web.UI.WebControls.WebParts;  using System.Xml.Linq;  namespace eventdemo  {  public partial class \_Default : System.Web.UI.Page  {  protected void Page\_Load(object sender, EventArgs e)  {  lblmessage.Text += "Page load event handled. <br />";  if (Page.IsPostBack)  {  lblmessage.Text += "Page post back event handled.<br/>";  }  }  protected void Page\_Init(object sender, EventArgs e)  {  lblmessage.Text += "Page initialization event handled.<br/>";  }  protected void Page\_PreRender(object sender, EventArgs e)  {  lblmessage.Text += "Page prerender event handled. <br/>";  }  protected void btnclick\_Click(object sender, EventArgs e)  {  lblmessage.Text += "Button click event handled. <br/>";  }  }  } |   Run the page. The label shows page load, page initialization and the page pre-render events. Click the button to see effect:  ASP.NET Event Example |
| **5** | **Server.Transfer**Server.Transfer method request and response |
|  |  |
|  | **Response.redirect**  Round Trip by Response.Redirect method   |  |  |  | | --- | --- | --- | |  | Response | Server | | Url | Response.Redirect() will send you to a new page, update the address bar and add it to the Browser History. On your browser you can click back. | Server.Transfer() does not change the address bar, we cannot hit back.One should use Server.Transfer() when he/she doesn’t want the user to see where he is going. Sometime on a "loading" type page. | | WebServer | It redirects the request to some plain HTML pages on our server or to some other web server. | It transfers current page request to another .aspx page on the same server. | | Roundtrip | It causes additional roundtrips to the server on each request. | It preserves server resources and avoids the unnecessary roundtrips to the server. | | Querystring and variable | It doesn’t preserve Query String and Form Variables from the original request. | It preserves Query String and Form Variables (optionally). | | Bookmark | It enables to see the new redirected URL where it is redirected in the browser and be able to bookmark. | It doesn’t show the real URL where it redirects the request in the users Web Browser. | |
|  |  |
| **6** | **Rich Web Controls**  **AdRotator Control**  Example:  **ASPX**  <form id="form1" runat="server">  <div>  <asp:AdRotator ID="AdRotator1" runat="server"  AdvertisementFile="~/advert.xml" />  </div>  </form>  **XML**  <?xml version="1.0" encoding="utf-8" ?>  <Advertisements>  <Ad>  <!-- The URL for the ad image -->  <ImageUrl>~/images/image1.gif</ImageUrl>  <!-- The URL the ad redirects the user to -->  <NavigateUrl>http://www.microsoft.com</NavigateUrl>  <!-- The alternate text for the image -->  <AlternateText>Visit Microsoft's Site</AlternateText>  <!-- The relative number of times this ad should appear -->  <!-- compared to the others -->  <Impressions>80</Impressions>  <!-- The topic of this ad (used for filtering) -->  <Keyword>ProductInfo</Keyword>  </Ad>  <Ad>  <ImageUrl>~/images/image2.gif</ImageUrl>  <NavigateUrl>http://www.microsoft.com/technet</NavigateUrl>  <AlternateText>Support for IT Professionals</AlternateText>  <Impressions>40</Impressions>  <Keyword>Support</Keyword>  </Ad>  <Ad>  <ImageUrl>~/images/microsoft.gif</ImageUrl>  <NavigateUrl>http://msdn.microsoft.com</NavigateUrl>  <AlternateText>Support for developers</AlternateText>  <Impressions>40</Impressions>  <Keyword>Support</Keyword>  </Ad>  </Advertisements>  **Calendar Control**  Example:  **ASPX**  <form>  <div>  <asp:Calendar ID="Calendar1" runat="server" BackColor="White"  BorderColor="#3366CC" BorderWidth="1px" CellPadding="1"  DayNameFormat="Shortest" Font-Names="Verdana" Font-Size="8pt"  ForeColor="#003399" Height="200px" Width="220px" FirstDayOfWeek="Monday"  **ondayrender="Calendar1\_DayRender"**  **onselectionchanged="Calendar1\_SelectionChanged">**  <DayHeaderStyle BackColor="#99CCCC" ForeColor="#336666" Height="1px" />  <NextPrevStyle Font-Size="8pt" ForeColor="#CCCCFF" />  <OtherMonthDayStyle ForeColor="#999999" />  <SelectedDayStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />  <SelectorStyle BackColor="#99CCCC" ForeColor="#336666" />  <TitleStyle BackColor="#003399" BorderColor="#3366CC" BorderWidth="1px"  Font-Bold="True" Font-Size="10pt" ForeColor="#CCCCFF" Height="25px" />  <TodayDayStyle BackColor="#99CCCC" ForeColor="White" />  <WeekendDayStyle BackColor="#CCCCFF" />  </asp:Calendar>  <asp:Label ID="lblDate" runat="server"></asp:Label>  </div>  </form>  **CS**  protected void Calendar1\_SelectionChanged(object sender, EventArgs e)  {  lblDate.Text = Calendar1.SelectedDate.ToLongDateString();  }  protected void Calendar1\_DayRender(object sender, DayRenderEventArgs e)  {  if (e.Day.IsWeekend)  e.Day.IsSelectable = false;  if (e.Day.DayNumberText == "1")  e.Cell.BackColor = System.Drawing.Color.Red;  }  **Treeview Control**  Example:  **ASPX:**  <asp:Button ID=”btnGenTreeView” runat=”server” OnClick= ” btnGenTreeView\_Click” />  **CS:**  protected void btnGenTreeView \_Click(object sender, EventArgs e)  {  TreeView view = new TreeView();  TreeNode parent = new TreeNode();  parent.Text = "Parent";  TreeNode child = new TreeNode();  child.Text = "Child";  TreeNode grandchild = new TreeNode();  grandchild.Text = "GrandChild";  child.ChildNodes.Add(grandchild);  parent.ChildNodes.Add(child);  view.Nodes.Add(parent);  Form.Controls.Add(view);  }  **Menu Control**  Example:  **ASPX**  <form id="form1" runat="server">  <div>  <asp:Menu ID="Menu1" runat="server" Orientation="Horizontal"  onmenuitemclick="Menu1\_MenuItemClick">  <Items>  <asp:MenuItem Text="Tab 1" Value="0" Selected="true"></asp:MenuItem>  <asp:MenuItem Text="Tab 2" Value="1"></asp:MenuItem>  <asp:MenuItem Text="Tab 3" Value="2"></asp:MenuItem>  </Items>  </asp:Menu>  <asp:MultiView ID="multiview1" ActiveViewIndex="0" runat="server">  <asp:View ID="view1" runat="server">  This is first view<br />  This is first view<br />  This is first view<br />  This is first view<br />  </asp:View>  <asp:View ID="view2" runat="server">  This is second view<br />  This is second view<br />  This is second view<br />  This is second view<br />  </asp:View>  <asp:View ID="view3" runat="server">  This is third view<br />  This is third view<br />  This is third view<br />  This is third view<br />  </asp:View>  </asp:MultiView>  </div>  </form>  **CS**  protected void Menu1\_MenuItemClick(object sender, MenuEventArgs e)  {  Int32 index = Convert.ToInt32(e.Item.Value);  multiview1.ActiveViewIndex = index;  } |