In ASP.NET, standard controls are pre-built user interface elements that you can use to create web applications. These controls are part of the ASP.NET framework and provide a convenient way to design and develop web pages. Here are some of the commonly used standard controls in ASP.NET:

1. **Label Control**: The Label control is used to display static text on a web page. It's often used for headings, descriptions, or other non-editable text.

htmlCopy code

<asp:Label ID="lblMessage" runat="server" Text="Hello, World!" />

1. **TextBox Control**: The TextBox control allows users to input text. It's used for user input forms.

htmlCopy code

<asp:TextBox ID="txtName" runat="server" />

1. **Button Control**: The Button control represents a clickable button element that can trigger server-side events.

htmlCopy code

<asp:Button ID="btnSubmit" runat="server" Text="Submit" OnClick="btnSubmit\_Click" />

1. **DropDownList Control**: DropDownList is used for creating dropdown menus or select lists.

htmlCopy code

<asp:DropDownList ID="ddlCountries" runat="server"> <asp:ListItem Text="USA" Value="1" /> <asp:ListItem Text="Canada" Value="2" /> <asp:ListItem Text="UK" Value="3" /> </asp:DropDownList>

1. **RadioButtonList Control**: RadioButtonList is used to create a list of radio buttons where users can select a single option.

htmlCopy code

<asp:RadioButtonList ID="rblGender" runat="server"> <asp:ListItem Text="Male" Value="Male" /> <asp:ListItem Text="Female" Value="Female" /> </asp:RadioButtonList>

1. **CheckBox Control**: CheckBox allows users to select one or more options from a list of choices.

htmlCopy code

<asp:CheckBox ID="chkSubscribe" runat="server" Text="Subscribe to newsletter" />

1. **GridView Control**: GridView is used to display tabular data. It provides features like sorting, paging, and editing.

htmlCopy code

<asp:GridView ID="gvProducts" runat="server"> <!-- Columns and data binding defined in code-behind --> </asp:GridView>

1. **Repeater Control**: Repeater is used to display a repeating list of items with custom HTML formatting.

htmlCopy code

<asp:Repeater ID="rptProducts" runat="server"> <ItemTemplate> <div><%# Eval("ProductName") %></div> </ItemTemplate> </asp:Repeater>

1. **Image Control**: The Image control displays an image on a web page.

htmlCopy code

<asp:Image ID="imgLogo" runat="server" ImageUrl="~/Images/logo.png" />

1. **HyperLink Control**: HyperLink is used to create hyperlinks on a web page.

htmlCopy code

<asp:HyperLink ID="hypGoogle" runat="server" Text="Google" NavigateUrl="https://www.google.com" />

These are just a few examples of standard controls in ASP.NET. ASP.NET provides a rich set of controls that you can use to build interactive and dynamic web applications. Each control has properties, methods, and events that you can customize and interact with in your code-behind files to create responsive web applications.

**validation controls in asp.net**

In ASP.NET, validation controls are a set of server-side controls that help you validate user input on a web page before it is processed on the server. These controls ensure that the data entered by users is accurate and meets specific criteria. Here are some of the commonly used validation controls in ASP.NET:

1. **RequiredFieldValidator**: This control ensures that a user must enter data in an associated input control, such as a TextBox, CheckBox, or DropDownList.

htmlCopy code

<asp:TextBox ID="txtUsername" runat="server"></asp:TextBox> <asp:RequiredFieldValidator ID="rfvUsername" runat="server" ControlToValidate="txtUsername" ErrorMessage="Username is required" Display="Dynamic"> </asp:RequiredFieldValidator>

1. **RegularExpressionValidator**: This control checks if the user's input matches a specified regular expression pattern. It's commonly used for validating email addresses, phone numbers, or other custom formats.

htmlCopy code

<asp:TextBox ID="txtEmail" runat="server"></asp:TextBox> <asp:RegularExpressionValidator ID="revEmail" runat="server" ControlToValidate="txtEmail" ErrorMessage="Invalid email format" ValidationExpression="\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*"> </asp:RegularExpressionValidator>

1. **CompareValidator**: This control compares the value entered in one input control with the value in another input control. It's often used for password confirmation.

htmlCopy code

<asp:TextBox ID="txtPassword" runat="server" TextMode="Password"></asp:TextBox> <asp:TextBox ID="txtConfirmPassword" runat="server" TextMode="Password"></asp:TextBox> <asp:CompareValidator ID="cvPassword" runat="server" ControlToValidate="txtConfirmPassword" ControlToCompare="txtPassword" ErrorMessage="Passwords do not match"> </asp:CompareValidator>

1. **RangeValidator**: This control checks if the user's input falls within a specified range of values, such as a minimum and maximum value for a numeric input.

htmlCopy code

<asp:TextBox ID="txtAge" runat="server"></asp:TextBox> <asp:RangeValidator ID="rvAge" runat="server" ControlToValidate="txtAge" MinimumValue="18" MaximumValue="99" Type="Integer" ErrorMessage="Age must be between 18 and 99"> </asp:RangeValidator>

1. **CustomValidator**: CustomValidator allows you to write your own custom validation logic using server-side or client-side code.

htmlCopy code

<asp:TextBox ID="txtCustom" runat="server"></asp:TextBox> <asp:CustomValidator ID="cvCustom" runat="server" ControlToValidate="txtCustom" ClientValidationFunction="customValidation" OnServerValidate="serverValidation" ErrorMessage="Invalid input"> </asp:CustomValidator>

1. **ValidationSummary**: ValidationSummary control displays a summary of validation errors on the page.

htmlCopy code

<asp:ValidationSummary ID="vsSummary" runat="server" />

These validation controls work in conjunction with other ASP.NET server controls to ensure that user input is valid before it's processed on the server. They help improve the quality of data submitted by users and provide a better user experience by giving feedback on validation errors.

State Management in ASP.NET

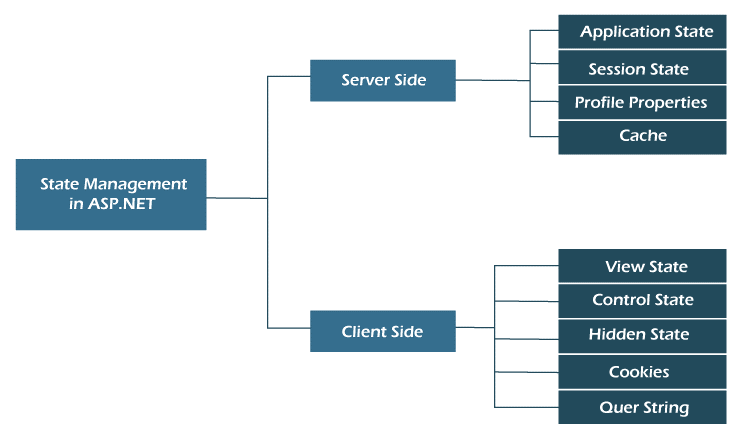
In this article, we will understand state management in ASP.NET in detail.

**What do you mean by State Management?**

State Management is a process by which state and page information is maintained over multiple requests for same or different pages. As HTTP is a stateless protocol, server does not store any information once the

response is sent back to client based on his request. When user submits request again, the server treats it as a new user. This is called stateless model. This model was workable when static web sites were developed and hosted in the past. Now, with interactive web sites or dynamic web site, there is a need to preserve some information to identify user, interact with user again and again within same session and same application. This concept is known as Stateful protocol. The information can be related to user, data objects, web pages or server objects.

To support this kind of model, ASP.NET provides two types of State Management techniques, **server side and client side** as shown in Figure.



Server Side State Management Options

ASP.NET provides facility to save information on server side as well as in client side. The following options are available in Server Side State Management.

* **Application State:** Application state allows saving of data at application level which is accessible throughout the life of an application. The life of application starts when IIS is started and ends when IIS is stopped.
* **Session State:** The session state persists till the user is active or session time expires. When a user submits request, a session is started. The session gets over either when time expires or user explicitly abandons the sessions. The required information can be saved in session for later user by same user within the same application.
* **Profile Properties:** This option also allows saving of user specific data. It is similar to Session state except the data stored in Profile state never expires on use this property, the SQLProfileProvider class needs to be configured. It allows storing of data in SQL database. Since data is stored in database and not in application memory, therefore there is no risk of losing data even if IIS is restarted again and again.
* **Cache:** Caching is a technique by which frequently used data and web pages are stored in cache so that repeated cost of retrieval can be avoided. Storing frequently used data in cache ensures high availability, improved performance and scalability. Cache is object of System.Web.Caching Cache class. The main disadvantage of using Cache is that it is unreliable. The previously stored data stored in cache is deleted automatically to meet memory requirement of current process.

Client Side State Management Options

The options available in client side state management help in storing information either in the page or at the client computer. No information is stored at server side. The followings options are used for client side state management.

* **View State:** View state provides facility to preserve page and values of controls at the client side. The information is stored after post back. **Post back** is a request from user for the page which is not for the first time. If value of **IsPostBack property** is true, it means page is not requested for the first time. The view state can be at page level, application level, machine level and control level. In page level state management, as long as the user is on current page, the information is retained. Whenever user submits form, the current state of page and controls are hashed into a string and saved in hidden field of the page. More than one hidden field can be used if data exceed limit set by **MaxPageStateFieldLength property**. When page is sent to server, the page parses the view state string and restores the information. This is default mechanism. The view state can be disabled at any stage. The property **EnableViewState="false"** is used in code when it is required to disable view state

The demonstrate the concept of **view state** option, consider an ASP.NET project haring one ben Each time a button is clicked, it displays the number of times the button is clicked.

* **Control State:** This is another client side state management option. This is used when there is a need to store control data related to Custom control. View state can be disabled but control state cannot be disabled.
* **Hidden Field State:** ASP.NET allows storing information in a hidden field which is a server control and can be used to store information at page level. The value of the hidden field is sent to HTTP form collection along with value of other controls. The hidden file can be created in source file as given below.<input type="hidden" id="username" name="username" value=""

This hidden field can be accessed in code behind file as given below. Dim st as String = Request QueryString("username")

* **Cookies:** Cookie is a small amount of information that is stored in client machine.
* **QueryString:** A QueryString contains the information that is sent to server with URL.

# ASP.NET Page Lifecycle

In ASP.NET, a web page has execution lifecycle that includes various phases. These phases include initialization, instantiation, restoring and maintaining state etc. it is required to understand the page lifecycle so that we can put custom code at any stage to perform our business logic.

## Page Lifecycle stages

The following table contains the lifecycle stages of ASP.NET web page.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Page request | This stage occurs before the lifecycle begins. When a page is requested by the user, ASP.NET parses and compiles that page. |
| Start | In this stage, page properties such as Request and response are set. It also determines the Request type. |
| Initialization | In this stage, each control's UniqueID property is set. Master page is applied to the page. |
| Load | During this phase, if page request is postback, control properties are loaded with information. |
| Postback event handling | In this stage, event handler is called if page request is postback. After that, the Validate method of all validator controls is called. |
| Rendering | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property. |
| Unload | At this stage the requested page has been fully rendered and is ready to terminate.at this stage all properties are unloaded and cleanup is performed. |

A requested page first loaded into the server memory after that processes and sent to the bowser. At last it is unloaded from the server memory. ASP.NET provides methods and events at each stage of the page lifecycle that we can use in our application. In the following table, we are tabled events.

## ASP.NET Life Cycle Events

|  |  |
| --- | --- |
| **Page Event** | **Typical Use** |
| PreInit | This event is raised after the start stage is complete and before the initialization stage. |
| Init | This event occurs after all controls have been initialized. We can use this event to read or initialize control properties. |
| InitComplete | This event occurs at the end of the page's initialization stage. We can use this event to make changes to view state that we want to make sure are persisted after the next postback. |
| PreLoad | This event is occurs before the post back data is loaded in the controls. |
| Load | This event is raised for the page first time and then recursively for all child controls. |
| Control events | This event is used to handle specific control events such as Button control' Click event. |
| LoadComplete | This event occurs at the end of the event-handling stage. We can use this event for tasks that require all other controls on the page be loaded. |
| PreRender | This event occurs after the page object has created all controls that are required in order to render the page. |
| PreRenderComplete | This event occurs after each data bound control whose DataSourceID property is set calls its DataBind method. |
| SaveStateComplete | It is raised after view state and control state have been saved for the page and for all controls. |
| Render | This is not an event; instead, at this stage of processing, the Page object calls this method on each control. |
| Unload | This event raised for each control and then for the page. |