**ASP.Net Cheat Sheet**

**Custom Control:**

<https://www.youtube.com/watch?v=UZkc-4R5lCw>

<https://www.tutorialspoint.com/asp.net/asp.net_custom_controls.htm>

A control that at first is invisible/inactive in the UI, and can only be viewed after the guest is doing something. For example, a **Custom Control** can be an image of a calendar and once the guest is clicking it, they are presented with a list of dates. Once the guest selected a dated, it should fill the corresponding text box, and list of dates returned to be hidden. They are deployed as individual assemblies. They are compiled into a Dynamic Link Library (DLL) and used as any other ASP.NET server control. They could be created in either of the following way:

1. By deriving a custom control from an existing control
2. By composing a new custom control combing two or more existing controls.
3. By deriving from the base control class.

**User Controls (.ascx):**

<https://learn.microsoft.com/en-us/previous-versions/aspnet/fb3w5b53(v=vs.100)>

<https://www.c-sharpcorner.com/UploadFile/sourabh_mishra1/usercontrol-in-Asp-Net/>

<https://www.c-sharpcorner.com/interview-question/difference-between-custom-control-and-user-control>

An ASP.NET Web user control is similar to a complete ASP.NET Web page (.aspx file), with both a user interface page and code. You create the user control in much the same way you create an ASP.NET page and then add the markup and child controls that you need. A user control can include code to manipulate its contents like a page can, including performing tasks such as data binding.

**Session:**

<https://www.javatpoint.com/asp-net-session>

<https://learn.microsoft.com/en-us/aspnet/web-api/overview/advanced/http-cookies>

In ASP.NET session is a state that is used to store and retrieve values of a user. It helps to identify requests from the same browser during a time period **(session)**. It is used to store value for the particular time **session**. By default, **ASP.NET session** state is enabled for all ASP.NET applications. Each created session is stored in SessionStateItemCollection object. We can get current session value by using Session property of Page object.

**View State:**

<https://learn.microsoft.com/en-us/previous-versions/aspnet/bb386448(v=vs.100)>

<https://www.c-sharpcorner.com/UploadFile/225740/what-is-view-state-and-how-it-works-in-Asp-Net53/>

<https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/view-state-in-asp-dot-net>

The method that the ASP.NET page framework uses to preserve page and control values between round trips. When the HTML markup for the page is rendered, the current state of the page and values that must be retained during postback are serialized into base64-encoded strings. This information is then put into the view state hidden field or fields. Basically, a web application is stateless, and as such, whenever we make a request to the server to get the page, it is getting lost immediately along with all instances created as part of the page rendering, as soon as we leave it/making a new request. As such, we use a state management techniques to preserve the instances of the page, and we make the preservation in a **View State**.

**Application State:**

<https://learn.microsoft.com/en-us/previous-versions/aspnet/ms178594(v=vs.100)>

Application state is a data repository available to all classes in an ASP.NET application. Application state is stored in memory on the server and is faster than storing and retrieving information in a database. Unlike session state, which is specific to a single user session, application state applies to all users and sessions. Therefore, application state is a useful place to store small amounts of often-used data that does not change from one user to another.

**Event:**

<https://www.tutorialspoint.com/asp.net/asp.net_event_handling.htm>

An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events. For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it. Events in ASP.NET raised at the client machine, and handled at the server machine. 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. If it has, the event handler is executed.

**Event Handles for UI Components:**

<https://www.mygreatlearning.com/asp-net/tutorials/asp-net-event-handling>

ASP.NET provides us the feature to handle the events in web Form. It allows us to implement event-based in our application. Suppose we have a button on our web page and we have to add action to that button. We can easily do this by writing a click event for the button. This can be done on both client and server-side. In ASP.NET web form, events associated with server control originates on the client-side but are resolved on the webserver by ASP.NET.

**ASP.Net Page Life Cycle:**

<https://learn.microsoft.com/en-us/previous-versions/aspnet/ms178472(v=vs.100)>

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Page request | The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page. |
| Start | In the start stage, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. The page also sets the UICulture property. |
| Initialization | During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state. |
| Load | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |
| Postback Event Handling | If the request is a postback, control event handlers are called. After that, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.) |
| 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 | The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed. |