# State Management

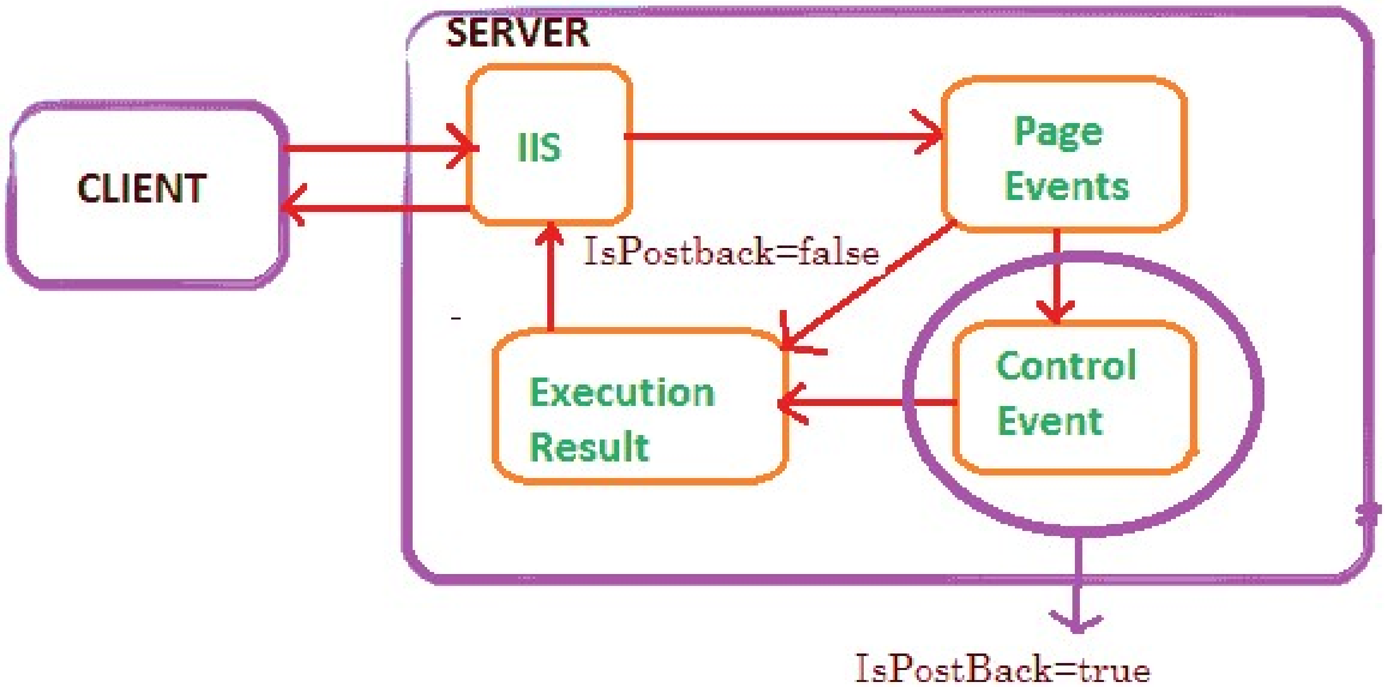
## Introduction

- State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless,
  - i.e., by default, for each page posted to the server, the state of controls is lost.
- Stateless means, whenever we visit a website, our browser communicates with the respective server depending on our requested functionality or the request.
- The browser communicates with the respective server using the HTTP or HTTPs protocol.
- Nowadays all web apps demand a high level of state management from control to application level.

## IsPostBack

- IsPostBack determines whether a form is posted to the page or not.
- Every time you invoke a server control that has a postback functionality, the page is refreshed.
- Postback is actually sending all the information from client to web server, then web server process all those contents and returns back to the client.
- Most of the time ASP control will cause a post back (e. g. buttonclick) but some don't unless you tell them to do In certain events ( Listbox Index Changed, RadioButton Checked etc..) in an ASP.NET page upon which a PostBack might be needed.

# IsPostBack


SERVER
CLIENT
IIS
Page Events
IsPostback=false
Control Event
Execution Result
IsPostBack=true

## IsPostBack

- IsPostBack is a property of the Asp.Net page that tells if the page is on its initial load or user has performed a button on the web page that has caused the page to post back to itself.
- The value of the Page.IsPostBack property will be set to true when the page is executing after a postback, and false otherwise.
- We can check the value of this property based on the value and we can populate the controls on the page.
- **Is Postback is normally used on page _load event to detect if the web page is getting generated due to postback requested by a control on the page or if the page is getting loaded for the first time.**

## Why to maintain States?

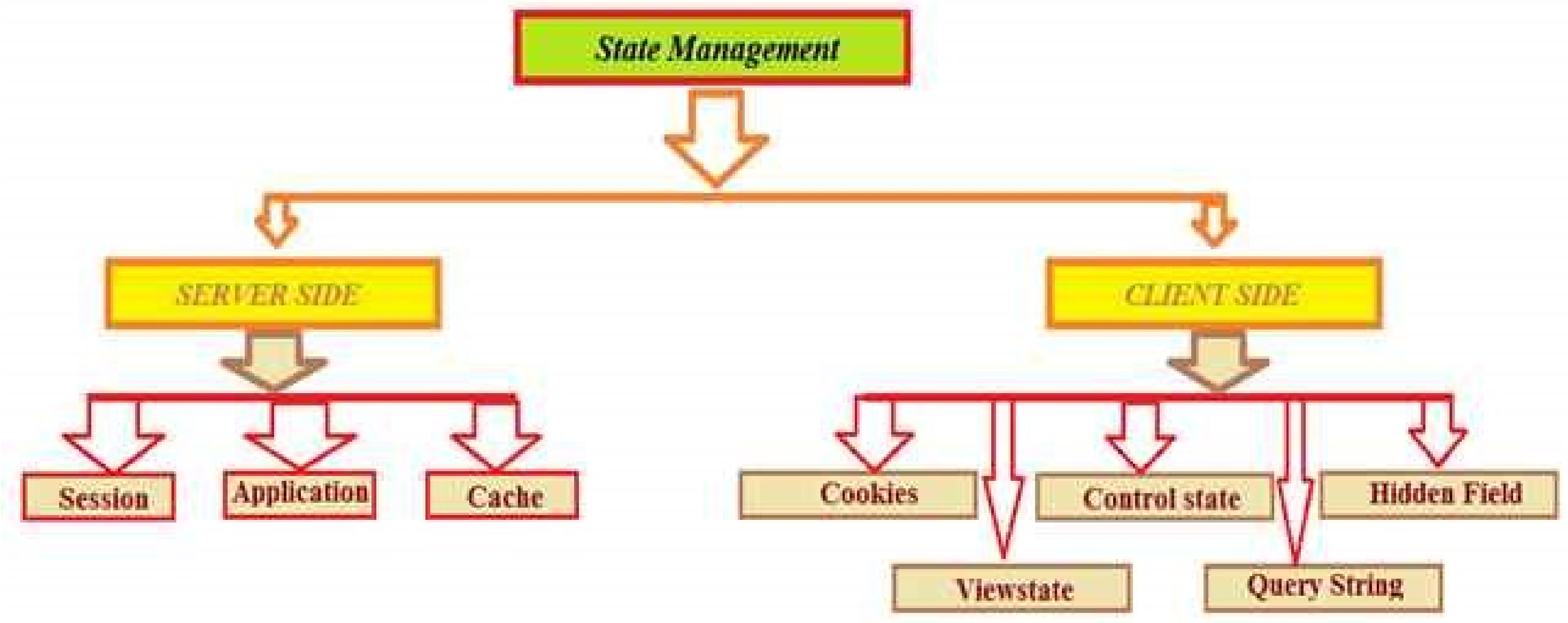
- A new instance of the Web page class is created each time the page is posted to the server. If a user enters information into a web application, that information would be lost in the round trip from the browser (MSDN).
- State management maintains and stores the information of any user till the end of the user session.

## Levels of State Management

- Control level: In ASP.NET, by default controls provide state management automatically.
- Variable or object level: In ASP.NET, member variables at page level are stateless and thus we need to maintain state explicitly.
- Single or multiple page level: State management at single as well as multiple page level i.e., managing state between page requests.
- User level: State should be preserved as long as a user is running the application.
- Application level: State available for complete application irrespective of the user, i.e., should be available to all users.
- Application to application level: State management between or among two or more applications.

# State Management Part

- Two types of State Management techniques are available in ASP.NET as in figure 1.

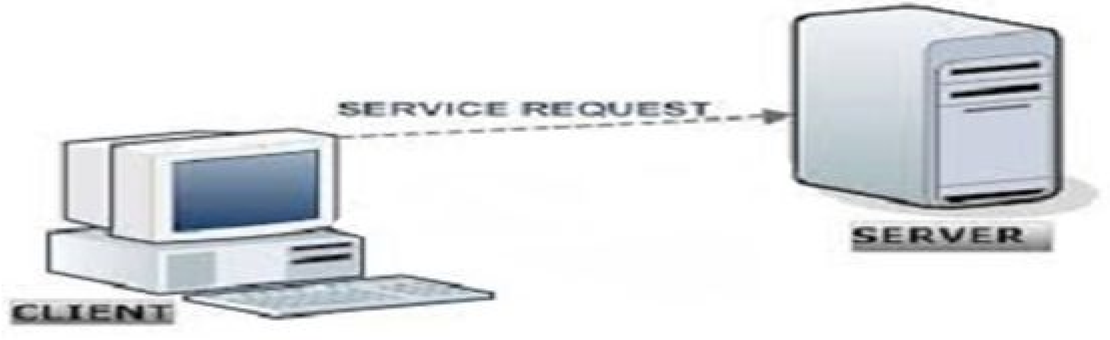


## Types of State Management

- There are two types of state management techniques: client side and server side.
- **Client side**
  - Hidden Field
  - View State
  - Cookies
  - Control State
  - Query Strings
- **Server side**
  - Session (In-Proc, SQL Server, State Server)
  - Application
  - Cache

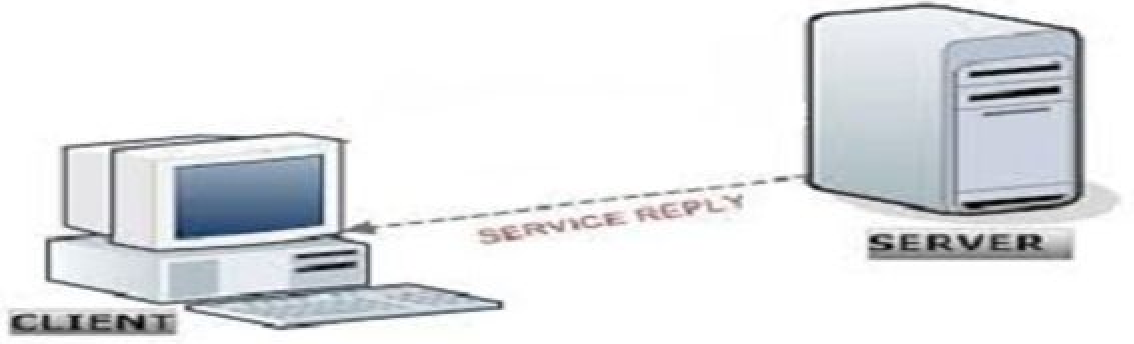
## Client-Side State Management

- Whenever we use Client-Side State Management, the state related information will directly get stored on the client-side.
- That specific information will travel back and communicate with every request generated by the user then afterwards provides responses after server-side communication.
- This architecture is something like the following,



## Server-Side State Management

- Server-Side State Management is different from Client-Side State Management but the operations and working is somewhat the same in terms of functionality.
- In Server-Side State Management all the information is stored in the server's memory.
- Due to this functionality there is more secure domains at the server side in comparison to Client-Side State Management.
- The structure is something like the following,



## Server-Side State Management

- **Session**
- Session is used to store user's information and/or uniquely identify a user (or say browser) by using a session ID.
- When users makes a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.
- Session is used to store information and identity. The server stores information using Sessionid.

## Server-Side State Management

- **Foe example:**

```
protected void btnSubmit_Click(object sender, EventArgs e)  {
    Session["UserName"] = txtName.Text;
    Response.Redirect("Home.aspx");
}
```

## Session Event

- Session event can be seen in project Global.asax file. Two types of Session Events
- **Session_Start**
- The Session_start event is raised every time a new users requests without a session ID. E.g.

```
void Session_Start(object sender, EventArgs e)  {
   Session["master"] = "~/Master.master";
}
```

- **Session_End**
- The Session_End event is raised when session is ended by a user or a time out using Session end method. E.g.

```
void Session_End(object sender, EventArgs e)  {
      Response.Write("Session_End");
}
```

## Storing Session Information

- *InProcMode*
  - In proc mode is the default mode provided by ASP.NET. In this mode, session values are stored in the web server's memory (in IIS).
  - If there are more than one IIS servers then session values are stored in each server separately on which request has been made.
  - Since the session values are stored in server, whenever server is restarted the session values will be lost.
  - E.g.

```
<configuration>
<sessionstate mode="InProc" cookieless="false" timeout="10"
    stateConnectionString="tcpip=127.0.0.1:80808" sqlConnectionString="Data
    Source=.\SqlDataSource;User ID=userid;Password=password"/>
</configuration>
```

## Storing Session Information

- *State Server Mode*
  - In this mode session data is stored in separate server.
  - This mode could store session in the web server but out of the application pool.
  - But usually if this mode is used there will be a separate server for storing sessions, i.e., stateServer.
  - The benefit is that when IIS restarts the session is available. It stores session in a separate Windows service.
  - For State server session mode, we have to configure it explicitly in the web config file and start the aspnet _state service. E.g.

```
<configuration>
<sessionstate mode="stateserver" cookieless="false" timeout="10"
   stateConnectionString="tcpip=127.0.0.1:42424"
   sqlConnectionString="Data Source=.\SqlDataSource;User ID =
   userid;Password=password"/>
</configuration>
```

## Storing Session Information

- *SQL Server Mode*
- Session is stored in a SQL Server database.
- This kind of session mode is also separate from IIS, i.e., session is available even after restarting the IIS server.
- This mode is highly secure and reliable but also has a disadvantage that there is overhead from serialization and deserialization of session data.
- This mode should be used when reliability is more important than performance. E.g.

```
<configuration>
<sessionstate mode="sqlserver" cookieless="false" timeout="10"
    stateConnectionString="tcpip=127.0.0.1:4 2424" sqlConnectionString="Data
    Source=.\SqlDataSource;User ID=userid;Password=password"/>
</configuration>
```

# Storing Session Information

- *Custom Mode*
  - Session data is stored in InProc, Sql Server, State server, etc. If you store session data with other new techniques then provide ASP.NET.
  - This way we have to maintain everything customized even generating session ID, data store, and also security.

| **Attributes** | **Description** |
|---|---|
| Cookieless true/false | Indicates that the session is used with or without cookie. cookieless set to true indicates sessions without cookies is used and cookieless set to false indicates sessions with cookies is used. cookieless set to false is the default set. |
| timeout | Indicates the session will abound if it is idle before session is abounded explicitly (the default time is 20 min). |
| StateConnectionString | Indicates the session state is stored on the remote computer (server). This attribute is required when session mode is StateServer |
| SqlConnectionString | Indicates the session state is stored in the database. This attribute is required when session mode is SqlServer. |

## Server-Side State Management

- **Application**
- Application state is a server side state management technique.
- Application state is a global storage mechanism that is used to store data that needs to be available to all users of an ASP.NET application. Application state is stored in memory on the server, and it is specific to an ASP.NET application.It is also called application level state management. Data stored in the application should be of small size.
- How to get and set a value in the application object:

```
Application["Count"] = Convert.ToInt32(Application["Count"]) + 1;
//Set Value to The Application Object
Label1.Text = Application["Count"].ToString();
//Get Value from the Application Object
```

## Server-Side State Management

- To provide for the use of application state, ASP.NET creates an application state object for each application from the HTTPApplicationState class and stores this object in server memory.
- This object is represented by class file **global.asax.**
- Application State is mostly used to store **hit counters** and other **statistical data, global application data like tax rate, discount rate etc**. and to keep the track of users visiting the site.
- The **HttpApplicationState** class has the following properties:
- <u>Item(name): </u>The value of the application state item with the specified name. This is the default property of the HttpApplicationState class.
- <u>Count:</u> The number of items in the application state collection.

## Server-Side State Management

- The HttpApplicationState class has the following methods:
- Add(name, value): Adds an item to the application state collection.
- Clear: Removes all the items from the application state collection.
- Remove(name): Removes the specified item from the application state collection.
- RemoveAll: Removes all objects from an HttpApplicationState collection.
- RemoveAt: Removes an HttpApplicationState object from a collection by index.
- Lock(): Locks the application state collection so only the current user can access it.
- Unlock():Unlocks the application state collection so all the users can access it.

## Server-Side State Management

- Application state data is generally maintained by writing handlers for the events:
  - Application_Start
  - Application_End
  - Application_Error
  - Session_Start
  - Session_End

## Server-Side State Management

- **Cache**
- Cache is stored on server side. It implements Page Caching and data caching. Cache is use to set expiration polices

  *Response.Cache.SetExpiresTime(DateTime.Now.AddDays(1));*
- The Cache object is an instance of the System.Web.Caching.Cache class, stored on the serve as ASP.NET removes objects if the memory becomes scarce.
- Cache objects can have expiration polices set on them and is shared across users.
- You can implement Page Caching and Data Caching.

# Server-Side State Management

- **Profile Properties**
- Profile properties are a per-user storage mechanism that is used to store data that is specific to a user's profile.
- Profile properties are stored in a database, and they can be used to store data such as user preferences, contact information, and so on.
- To access a profile property, you can use the following code:

  *string profileProperty = (string)Profile["userName"];*
- To store a value in a profile property, you can use the following code:

  *Profile["userName"] = "John Smith";*

## Client-Side State Management

- **Hidden Field**
- Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client.
- It store one value for the variable and it is a preferable way when a variable's value is changed frequently.
- Hidden field control is not rendered to the client (browser) and it is invisible on the browser.
- A hidden field travels with every request like a standard control’s value.

## Client-Side State Management

- **Hidden Field**
- Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client.
- It store one value for the variable and it is a preferable way when a variable's value is changed frequently.
- Hidden field control is not rendered to the client (browser) and it is invisible on the browser.
- A hidden field travels with every request like a standard control’s value.

## Client-Side State Management

- **View State**
- The view state is the state of the page and all its controls.
- View State provides page level state management i.e., as long as the user is on the current page, state is available and the user redirects to the next page and the current page state is lost.
- View State can store any type of data because it is object type but it is preferable not to store a complex type of data due to the need for serialization and deserilization on each post back.
- View state is enabled by default for all server side controls of ASP.NET with a property EnableviewState set to true.

## Client-Side State Management

- **View State**
- When a page is sent back to the client, the changes in the properties of the page and its controls are determined, and stored in the value of a hidden input field named _VIEWSTATE.
- When the page is again posted back, the _VIEWSTATE field is sent to the server with the HTTP request.
- The view state could be enabled or disabled for:
  - **The entire application** by setting the EnableViewState property in the <pages> section of web.config file.
  - **A page** by setting the EnableViewState attribute of the Page directive, as <%@ Page Language="C#" EnableViewState="false" %>
  - **A control** by setting the Control.EnableViewState property. E.g.
    - TextBox1.EnableViewState=false;

## Client-Side State Management

- **View State**
- Data Objects That Can be Stored in View state
  - String
  - Boolean Value
  - Array Object
  - Array List Object
  - Hash Table
  - Custom type Converters
- **When We Should Use View State**
  - When the data to be stored is small.
  - Try to avoid secure data.

## Client-Side State Management

- **Difference between EnableViewState and ViewStateMode properties**
  - 1. Using EnableViewState property we only have 2 options We can turn off view state altogether, or Enable viewstate for the entire page and then turn it off on a control-by-control basis.
  - 2. If you want to turn of ViewState for the entire page and only enable it for specific controls on the page, then we have to use ViewStateMode property in conjunction with EnableViewState.
  - 3. EnableViewState property only accepts true or false values and the default value is true, where as ViewStateMode property can have a value of - Enabled, Disabled and inherit. Inherit is the default value for ViewStateMode property.
  - 4. ViewStateMode property is introduced in ASP.NET 4, where as EnableViewState exists from a long time.

## Client-Side State Management

- **Difference between EnableViewState and ViewStateMode properties**
  - 5. If EnableViewState is to True, only then the ViewStateMode settings are applied, where as, if EnableViewState is set to False then the control will not save its view state, regardless of the ViewStateMode setting.
  - In short if EnableViewState is set to False, ViewStateMode setting is not respected.
  - 6. To disable view state for a page and to enable it for a specific control on the page, set the EnableViewState property of the page and the control to true, set the ViewStateMode property of the page to Disabled, and set the ViewStateMode property of the control to Enabled.

## Client-Side State Management

- **Advantages of View State**
  - Easy to Implement.
  - No server resources are required: The View State is contained in a structure within the page load.
  - Enhanced security features: It can be encoded and compressed or Unicode implementation.
- **Disadvantages of View State**
  - <u>Security Risk:</u> The Information of View State can be seen in the page output source directly. You can manually encrypt and decrypt the contents, but It requires extra coding.
  - If security is a concern then consider using a Server-Based state Mechanism so that no sensitive information is sent to the client.
  - <u>Performance:</u> Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.
  - <u>Device limitation:</u> Mobile Devices might not have the memory capacity to store a large amount of View State data.
  - It can store values for the same page only.

# Client-Side State Management

- **View State Security**
- View State Data is stored in the form of Base 64 encoding but it is not more secure, anyone can easily break it. So there are the following 2 options:
- **Using the MAC for Computing the View State Hash Value**
- When the key is auto-generated then ASP.NET uses SHA-1 encoding to create a larger key. Those keys must be the same for all the server.
- If the key is not the same and the page is posted back to a different server than the one that created the page then the ASP.NET Page Framework raises an exception. We can enable it by using,
- <%Page Language="C#" EnableViewState="true“
- EnableViewStateMac="true";

## Client-Side State Management

- **Encryption**
- By using MAC Encoding we cannot prevent the viewing of the data so to prevent the viewing, transmit the page over SSL and encrypt the View State Data. To encrypt the data we have the ViewStateEncryptionMode Property and it has the following 3 options:
- Always: Encrypt the data Always.
- Never: Encrypt the data Never.
- Auto: Encrypt if  any Control request specially for Encryption

## Client-Side State Management

- **Cookies**
- Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser.
- It does not use server memory. Generally a cookie is used to identify users.
- Whenever a user makes a page request for the first time, the server creates a cookie and sends it to the client along with the requested page.
- Client browser receives that cookie and stores it on the client machine either permanently or temporarily (persistent or non persistence).
- The next time the user makes a request for the same site, either the same or another page, the browser checks the existence of the cookie for that site in the folder.
- If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.

## Client-Side State Management

- **Query String**
- Query string stores the value in URL. They store the value in the form of Key-Value pair. E.g.
  - Response.Redirect("ShowStringValue.aspx?Username=“+ txtUsername.Text);
- Query string is limited to simple string information, easily accessible and readable.
- These in a general case are used for holding some value from a different page and move these values to the different page.
- The information stored in it can be easily navigated to one page to another or to the same page as well.
- Uses real and virtual path values for URL routing.

## Client-Side State Management

- **Control State**
- Control State is another client side state management technique.
- Whenever we develop a custom control and want to preserve some information, we can use view state but suppose view state is disabled explicitly by the user, the control will not work as expected.
- For expected results for the control we have to use Control State property. Control state is separate from view state.
- **How to use control state property**: Control state implementation is simple. First override the OnInit() method of the control and add a call for the Page.RegisterRequiresControlState() method with the instance of the control to register.
- Then override LoadControlState and SaveControlState in order to save the required state information.