State management is the process by which you maintain state and page information over multiple requests for the same or different pages.   
  
  
**Types of State Management**  
  
There are 2 types State Management:   
  
**1. Client – Side State Management**   
This stores information on the client's computer by embedding the information into a Web page, a uniform resource locator(url), or a cookie. The techniques available to store the state information at the client end are listed down below:   
  
a. View State – Asp.Net uses View State to track the values in the Controls. You can add custom values to the view state. It is used by the Asp.net page framework to automatically save the values of the page and of each control just prior to rendering to the page. When the page is posted, one of the first tasks performed by page processing is to restore view state.   
  
b. Control State – If you create a custom control that requires view state to work properly, you should use control state to ensure other developers don’t break your control by disabling view state.   
  
c. Hidden fields – Like view state, hidden fields store data in an HTML form without displaying it in the user's browser. The data is available only when the form is processed.   
  
d. Cookies – Cookies store a value in the user's browser that the browser sends with every page request to the same server. Cookies are the best way to store state data that must be available for multiple Web pages on a web site.   
  
e. Query Strings - Query strings store values in the URL that are visible to the user. Use query strings when you want a user to be able to e-mail or instant message state data with a URL.   
  
**2. Server – Side State Management**  
a. Application State - Application State information is available to all pages, regardless of which user requests a page.   
  
b. Session State – Session State information is available to all pages opened by a user during a single visit.   
  
Both application state and session state information is lost when the application restarts. To persist user data between application restarts, you can store it using profile properties.   
  
**Implementation Procedure**  
  
  
**Client – Side State Management:**   
  
***View State:***  
The ViewState property provides a dictionary object for retaining values between multiple requests for the same page. When an ASP.NET page is processed, the current state of the page and controls is hashed into a string and saved in the page as a hidden field. If the data is too long for a single field, then ASP.NET performs view state chunking (new in ASP.NET 2.0) to split it across multiple hidden fields. The following code sample demonstrates how view state adds data as a hidden form within a Web page’s HTML:

<input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE” value="/wEPDwUKMTIxNDIyOTM0Mg9kFgICAw9kFgICAQ8PFgIeBFRleHQFEzQvNS8yMDA2IDE6Mzc6MTEgUE1kZGROWHn/rt75XF/pMGnqjqHlH66cdw==" />

Encrypting of the View State: You can enable view state encryption to make it more difficult for attackers and malicious users to directly read view state information. Though this adds processing overhead to the Web server, it supports in storing confidential information in view state. To configure view state encryption for an application does the following: 

<Configuration>

<system.web>

<pages **viewStateEncryptionMode**="Always"/>

</system.web>

</configuration>

Alternatively, you can enable view state encryption for a specific page by setting the value in the page directive, as the following sample demonstrates: 

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" ViewStateEncryptionMode="Always"%>

View State is enabled by default, but if you can disable it by setting the EnableViewState property for each web control to false. This reduces the server processing time and decreases page size.   
Reading and Writing Custom View State Data:   
If you have a value that you’d like to keep track of while the user is visiting a single ASP.NET Web page, adding a custom value to ViewState is the most efficient and secure way to do that. However, ViewState is lost if the user visits a different Web page, so it is useful only for temporarily storing values.   
Example: Determine the time of last visit to the page 

// Check if View State object exists, and display it if it does

If (ViewState ["lastVisit"]!= null)

Label1.Text = (string)ViewState["lastVisit"]; else

Label1.Text = "lastVisit ViewState not defined.";

// Define the ViewState object for the next page view ViewState.Add("lastVisit", DateTime.Now.ToString());

***Control State:*** If you create a custom control that requires ViewState, you can use the ControlState property to store state information for your control. ControlState allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property. To use control state in a custom control, your control must override the OnInit method and call the Register-RequiresControlState method during initialization and then override the SaveControl-State and LoadControlState methods.   
  
***Hidden fields:***ViewState stores information in the Web page using hidden fields. Hidden fields are sent back to the server when the user submits a form; however, the information is never displayed by the Web browser (unless the user chooses to view the page source). ASP.NET allows you to create your own custom hidden fields and store values that are submitted with other form data. A HiddenField control stores a single variable in its Value property and must be explicitly added to the page. You can use hidden fields only to store information for a single page, so it is not useful for storing session data. If you use hidden fields, you must submit your pages to the server using Hypertext Transfer Protocol (HTTP) POST (which happens if the user presses a button) rather than requesting the page using HTTP GET (which happens if the user clicks a link). Unlike view state data, hidden fields have no built-in compression, encryption, hashing, or chunking, so users can view or modify data stored in hidden fields.   
  
***Cookies:*** Web applications can store small pieces of data in the client’s Web browser by using cookies. A cookie is a small amount of data that is stored either in a text file on the client file system (if the cookie is persistent) or in memory in the client browser session (if the cookie is temporary). The most common use of cookies is to identify a single user as he or she visits multiple Web pages.   
  
Reading and Writing Cookies:   
A Web application creates a cookie by sending it to the client as a header in an HTTP response. The Web browser then submits the same cookie to the server with every new request.   
Create a cookie -> add a value to the Response.Cookies HttpCookieCollection.   
Read a cookie -> read values in Request.Cookies.   
Example: 

// Check if cookie exists, and display it if it does

if (Request.Cookies["lastVisit"] != null) // Encode the cookie in case the cookie contains client-side script Label1.Text = Server.HtmlEncode(Request.Cookies["lastVisit"].Value);

else Label1.Text = "No value defined";

// Define the cookie for the next visit Response.Cookies["lastVisit"].Value = DateTime.Now.ToString();Response.Cookies["lastVisit"].Expires = DateTime.Now.AddDays(1);

If you do not define the Expires property, the browser stores it in memory and the cookie is lost if the user closes his or her browser.   
  
To delete a cookie, overwrite the cookie and set an expiration date in the past. You can’t directly delete cookies because they are stored on the client’s computer.   
Controlling the Cookie Scope: By default, browsers won’t send a cookie to a Web site with a different hostname. You can control a cookie’s scope to either limit the scope to a specific folder on the Web server or expand the scope to any server in a domain. To limit the scope of a cookie to a folder, set the Path property, as the following example demonstrates:   
  
Example: 

Response.Cookies["lastVisit"].Path = "/Application1";

Through this the scope is limited to the “/Application1” folder that is the browser submits the cookie to any page with in this folder and not to pages in other folders even if the folder is in the same server. We can expand the scope to a particular domain using the following statement:   
Example:   
Response.Cookies[“lastVisit”].Domain = “Contoso”;   
  
Storing Multiple Values in a Cookie:   
Though it depends on the browser, you typically can’t store more than 20 cookies per site, and each cookie can be a maximum of 4 KB in length. To work around the 20-cookie limit, you can store multiple values in a cookie, as the following code demonstrates:   
Example: 

Response.Cookies["info"]["visit"].Value = DateTime.Now.ToString();

Response.Cookies["info"]["firstName"].Value = "Tony";

Response.Cookies["info"]["border"].Value = "blue";

Response.Cookies["info"].Expires = DateTime.Now.AddDays(1);

Running the code in this example sends a cookie with the following value to the Web browser:   
(visit=4/5/2006 2:35:18 PM) (firstName=Tony) (border=blue)   
  
***Query Strings:***Query strings are commonly used to store variables that identify specific pages, such as search terms or page numbers. A query string is information that is appended to the end of a page URL. A typical query string might look like the following real-world example:   
http://support.microsoft.com/Default.aspx?kbid=315233   
In this example, the URL identifies the Default.aspx page. The query string (which starts with a question mark [?]) contains a single parameter named “kbid,” and a value for that parameter, “315233.” Query strings can also have multiple parameters, such as the following real-world URL, which specifies a language and query when searching the Microsoft.com Web site:   
http://search.microsoft.com/results.aspx?mkt=en-US&setlang=en-US&q=hello+world   
  
Value Name | ASP.NET Object | Value   
mkt | Request.QueryString[“mkt”] | en-US   
setlang | Request.QueryString[“setlang”] | en-US   
q | Request.QueryString[“q”] | hello world   
  
Limitations for Query Strings:   
1. Some Browsers and client devices impose a 2083 – character limit on the length of the URL.   
2. You must submit the page using an HTTP GET command in order for query string values to be available during page processing. Therefore, you shouldn’t add query strings to button targets in forms.   
3. You must manually add query string values to every hyperlink that the user might click.   
Example: 

Label1.Text = "User: " + Server.HtmlEncode(Request.QueryString["user"]) +

", Prefs: " + Server.HtmlEncode(Request.QueryString["prefs"]) +

", Page: " + Server.HtmlEncode(Request.QueryString["page"]);

**Server - Side State Management:**   
  
***Application State:***ASP.NET allows you to save values using application state, a global storage mechanism that is accessible from all pages in the Web application. Application state is stored in the Application key/value dictionary. Once you add your application-specific information to application state, the server manages it, and it is never exposed to the client. Application state is a great place to store information that is not user-specific. By storing it in the application state, all pages can access data from a single location in memory, rather than keeping separate copies of the data. Data stored in the Application object is not permanent and is lost any time the application is restarted.   
  
ASP.NET provides three events that enable you to initialize Application variables (free resources when the application shuts down) and respond to Application errors:   
  
a. Application\_Start: Raised when the application starts. This is the perfect place to initialize Application variables.   
  
b. Application\_End: Raised when an application shuts down. Use this to free application resources and perform logging.   
  
c. Application\_Error: Raised when an unhandled error occurs. Use this to perform error logging.   
  
  
***Session State:*** ASP.NET allows you to save values using session state, a storage mechanism that is accessible from all pages requested by a single Web browser session. Therefore, you can use session state to store user-specific information. Session state is similar to application state, except that it is scoped to the current browser session. If different users are using your application, each user session has a different session state. In addition, if a user leaves your application and then returns later after the session timeout period, session state information is lost and a new session is created for the user. Session state is stored in the Session key/value dictionary.   
  
You can use session state to accomplish the following tasks:   
i. Uniquely identify browser or client-device requests and map them to individual session instances on the server. This allows you to track which pages a user saw on your site during a specific visit.   
  
ii. Store session-specific data on the server for use across multiple browser or client-device requests during the same session. This is perfect for storing shopping cart information.   
  
iii. Raise appropriate session management events. In addition, you can write application code leveraging these events.   
  
ASP.NET session state supports several different storage options for session data:   
  
a. InProc Stores session state in memory on the Web server. This is the default, and it offers much better performance than using the ASP.NET state service or storing state information in a database server. InProc is fine for simple applications, but robust applications that use multiple Web servers or must persist session data between application restarts should use State Server or SQLServer.   
  
b. StateServer Stores session state in a service called the ASP.NET State Service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm. ASP.NET State Service is included with any computer set up to run ASP.NET Web applications; however, the service is set up to start manually by default. Therefore, when configuring the ASP.NET State Service, you must set the startup type to Automatic.   
  
c. SQLServer Stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm. On the same hardware, the ASP.NET State Service outperforms SQLServer. However, a SQL Server database offers more robust data integrity and reporting capabilities.   
  
d. Custom Enables you to specify a custom storage provider. You also need to implement the custom storage provider.   
  
e. Off Disables session state. You should disable session state if you are not using it to improve performance.   
  
**Advantages**  
  
  
**Advantages of Client – Side State Management:**   
  
1. Better Scalability: With server-side state management, each client that connects to the Web server consumes memory on the Web server. If a Web site has hundreds or thousands of simultaneous users, the memory consumed by storing state management information can become a limiting factor. Pushing this burden to the clients removes that potential bottleneck.   
  
2. Supports multiple Web servers: With client-side state management, you can distribute incoming requests across multiple Web servers with no changes to your application because the client provides all the information the Web server needs to process the request. With server-side state management, if a client switches servers in the middle of the session, the new server does not necessarily have access to the client’s state information. You can use multiple servers with server-side state management, but you need either intelligent load-balancing (to always forward requests from a client to the same server) or centralized state management (where state is stored in a central database that all Web servers access).   
  
**Advantages of Server – Side State Management:**   
  
1. Better security: Client-side state management information can be captured (either in transit or while it is stored on the client) or maliciously modified. Therefore, you should never use client-side state management to store confidential information, such as a password, authorization level, or authentication status.   
  
2. Reduced bandwidth: If you store large amounts of state management information, sending that information back and forth to the client can increase bandwidth utilization and page load times, potentially increasing your costs and reducing scalability. The increased bandwidth usage affects mobile clients most of all, because they often have very slow connections. Instead, you should store large amounts of state management data (say, more than 1 KB) on the server.

HTTP ( Hyper Text Transfer Protocol) is a stateless protocol. When the client disconnects from the server, the ASP.Net engine discards the page objects. This way each web application can scale up to serve numerous requests simultaneously without running out of server memory.

However, there need to be some technique to store the information between requests and to retrieve it when required. This information i.e., the current value of all the controls and variables for the current user in the current session is called the State.

ASP.Net manages four types of state:

1. View State
2. Control State
3. Session State
4. Application State

View State:

The View State is the state of the page and all its controls. It is automatically maintained across posts by the ASP.Net framework.

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 post 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.

It is implemented using a view state object defined by the StateBag class which defines a collection of view state items. The state bag is a data structure containing attribute/value pairs, stored as strings associated with objects.

The StateBag class has the following properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Item(name) | The value of the view state item with the specified name. This is the default property of the StateBag class |
| Count | The number of items in the view state collection |
| Keys | Collection of keys for all the items in the collection |
| Values | Collection of values for all the items in the collection |

The StateBag class has the following methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Add(name, value) | Adds an item to the view state collection and existing item is updated |
| Clear | Removes all the items from the collection |
| Equals(Object) | Determines whether the specified object is equal to the current object. |
| Finalize | Allows it to free resources and perform other cleanup operations. |
| GetEnumerator | Returns an enumerator that iterates over all the key/value pairs of the StateItem objects stored in the StateBag object. |
| GetType | Gets the Type of the current instance. |
| IsItemDirty | Checks a StateItem object stored in the StateBag object to evaluate whether it has been modified. |
| Remove(name) | Removes the specified item. |
| SetDirty | Sets the state of the StateBag object as well as the Dirty property of each of the StateItem objects contained by it. |
| SetItemDirty | Sets the Dirty property for the specified StateItem object in the StateBag object. |
| ToString | Returns a String representing the state bag object. |

**Example:**

The following example demonstrates the concept of storing view state. Let us keep a counter, which is incremented each time the page is post back by clicking a button on the page. A label control shows the value in the counter.

The markup file:

|  |
| --- |
| <%@ Page Language="C#"  AutoEventWireup="true"  CodeBehind="Default.aspx.cs"  Inherits="statedemo.\_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>  <h3>View State demo</h3>  Page Counter:  <asp:Label ID="lblCounter" runat="server" />  <asp:Button ID="btnIncrement" runat="server"  Text="Add Count"  onclick="btnIncrement\_Click" />  </div>  </form>  </body>  </html> |

The code behind file for the example is shown here:

|  |
| --- |
| public partial class \_Default : System.Web.UI.Page  {  public int counter  {  get  {  if (ViewState["pcounter"] != null)  {  return ((int)ViewState["pcounter"]);  }  else  {  return 0;  }  }  set  {  ViewState["pcounter"] = value;  }  }    protected void Page\_Load(object sender, EventArgs e)  {  lblCounter.Text = counter.ToString();  counter++;  }  } |

This would produce following result:



Control State:

Control state cannot be modified, accessed directly or disabled.

Session State:

When a user connects to an ASP.Net website, a new session object is created. When session state is turned on, a new session state object is created for each new request. This session state object becomes part of the context and it is available through the page.

Session state is generally used for storing application data like inventory or supplier list, or a customer record or shopping cart. It can also keep information about the user and his preference and keep track of pending operations.

Sessions are identified and tracked with a 120-bit SessionID, which is passed from client to server and back as cookie or a modified URL. The SessionID is globally unique and random.

The session state object is created from the HttpSessionState class, which defines a collection of session state items.

The HttpSessionState class has the following properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| SessionID | The unique session identifier |
| Item(name) | The value of the session state item with the specified name. This is the default property of the HttpSessionState class |
| Count | The number of items in the session state collection |
| TimeOut | Gets and sets the amount of time, in minutes, allowed between requests before the session-state provider terminates the session. |

The HttpSessionState class has the following methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Add(name, value) | Adds an item to the session state collection |
| Clear | Removes all the items from session state collection |
| Remove(name) | Removes the specified item from the session state collection |
| RemoveAll | Removes all keys and values from the session-state collection. |
| RemoveAt | Deletes an item at a specified index from the session-state collection. |

The session state object is a name-value pair, to store and retrieve some information from the session state object the following code could be used:

|  |
| --- |
| void StoreSessionInfo()  {  String fromuser = TextBox1.Text;  Session["fromuser"] = fromuser;  }  void RetrieveSessionInfo()  {  String fromuser = Session["fromuser"];  Label1.Text = fromuser;  } |

The above code stores only strings in the Session dictionary object, however, it can store all the primitive data types and arrays composed of primitive data types, as well as the DataSet, DataTable, HashTable, and Image objects, as well as any user defined class that inherits from the ISerializable object.

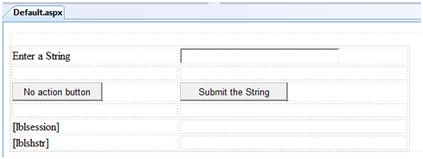
**Example:**

The following example demonstrates the concept of storing session state. There are two buttons on the page, a text box to enter string and a label to display the text stored from last session.

The mark up file:

|  |
| --- |
| <%@ Page Language="C#"  AutoEventWireup="true"  CodeFile="Default.aspx.cs"  Inherits="\_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>  &nbsp; &nbsp; &nbsp;  <table style="width: 568px; height: 103px">  <tr>  <td style="width: 209px">  <asp:Label ID="lblstr" runat="server"  Text="Enter a String" Width="94px">  </asp:Label>  </td>  <td style="width: 317px">  <asp:TextBox ID="txtstr" runat="server" Width="227px">  </asp:TextBox>  </td>  </tr>  <tr>  <td style="width: 209px"></td>  <td style="width: 317px"></td>  </tr>  <tr>  <td style="width: 209px">  <asp:Button ID="btnnrm" runat="server"  Text="No action button" Width="128px" />  </td>  <td style="width: 317px">  <asp:Button ID="btnstr" runat="server"  OnClick="btnstr\_Click" Text="Submit the String" />  </td>  </tr>  <tr>  <td style="width: 209px">  </td>  <td style="width: 317px">  </td>  </tr>  <tr>  <td style="width: 209px">  <asp:Label ID="lblsession" runat="server"  Width="231px">  </asp:Label>  </td>  <td style="width: 317px">  </td>  </tr>  <tr>  <td style="width: 209px">  <asp:Label ID="lblshstr" runat="server">  </asp:Label>  </td>  <td style="width: 317px">  </td>  </tr>  </table>  </div>  </form>  </body>  </html> |

It should look like the following in design view:



The code behind file is given here:

|  |
| --- |
| public partial class \_Default : System.Web.UI.Page  {  String mystr;  protected void Page\_Load(object sender, EventArgs e)  {  this.lblshstr.Text = this.mystr;  this.lblsession.Text = (String)this.Session["str"];  }  protected void btnstr\_Click(object sender, EventArgs e)  {  this.mystr = this.txtstr.Text;  this.Session["str"] = this.txtstr.Text;  this.lblshstr.Text = this.mystr;  this.lblsession.Text = (String)this.Session["str"];  }  } |

Run the file and observe how it works:



Application State

An ASP.Net application is the collection of all web pages, code and other files within a single virtual directory on a web server. When information is stored in application state, it is available to all the users.

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 track of users visiting the site.

The HttpApplicationState class has the following properties:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| Item(name) | The value of the application state item with the specified name. This is the default property of the HttpApplicationState class. |
| Count | The number of items in the application state collection. |

The HttpApplicationState class has the following methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| 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. |

Application state data is generally maintained by writing handlers for the events:

* Application\_Start
* Application\_End
* Application\_Error
* Session\_Start
* Session\_End

The following code snippet shows the basic syntax for storing application state information:

|  |
| --- |
| Void Application\_Start(object sender, EventArgs e)  {  Application["startMessage"] = "The application has started.";  }  Void Application\_End(object sender, EventArgs e)  {  Application["endtMessage"] = "The application has ended.";  } |

## tate Management Options

In a broader view there are two ways to manage web page’s **state**. It could be on Client-side and could be on Server-side. Both of these are used depending on the implications and requirements.

First we will look into the client side **state** **management** options.

Client-side **state** **management**: Under Client side **state** **management** no information is maintained on the server between round trips. Information will be stored in the page or on the client’s computer.

The following are the various ways in which Client side **state** can be maintained:

* Cookies
* Hidden Fields
* View **State**
* Query Strings

**Cookies:** A cookie is a small amount of data stored either in a text file on the client's file system or in-memory of the client browser session. Cookies are mainly used for tracking data settings. Although cookies are typically used to store user-specific configuration information and preferences, they can be used to store any client-specific **state** needed by an application (as long as that **state** is converted to string format) An example could be to customize the welcome page based on the user preferences.

The following is an example of using Cookies in **ASP.NET**:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

HttpCookie nameCookie= new HttpCookie("Name");

nameCookie.Value = TxtName.Text;

Response.Cookies.Add(nameCookie);

**Hidden Field:** A hidden field does not render visibly in the browser, but you can set its properties just as you can with a standard control. When a page is submitted to the server, the content of a hidden field is sent in the HTTP form collection along with the values of other controls. **ASP.NET** provides the HtmlInputHidden control that offers hidden field functionality.

Following is the code example in C#:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

protected System.Web.UI.HtmlControls.HtmlInputHidden Hidden1;

*//To set a value*

Hidden1.Value=”Test Value assigned to hidden field”;

*//to retrieve a value*

string strValue=Hidden1.Value;

Although its name is ‘Hidden’, its value is not hidden; you can see its value through ‘view -> source’ option in the browser.

**View State:** In addition to session **state** and cookie **state**, **ASP.NET** introduces the ability to store client-specific**state** through a mechanism called view **state**. View **state** is stored in a hidden field on each **ASP.NET** page called\_\_VIEW**STATE**. Each time a page is posted to itself, the contents of the \_\_VIEW**STATE** field are sent as part of the post. The primary use of view **state** is for controls to retain their **state** across post-backs.

The following is an example of using View**State** in C#:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

*//to save information*

View**State**.Add(“Test”,”View**State** Sample”);

*//to retrieve information*

string test=View**State**[“Test”];

**Query Strings:** Query strings provide a simple but limited way of maintaining some **state** information. You can easily pass information from one page to another, But most browsers and client devices impose a 255-character limit on the length of the URL. In addition, the query values are exposed to the Internet via the URL and so in some cases security may be an issue. A URL with query strings may look like this: *http://www.Himanshu.com/list.aspx?Age=27&Count=101*. When *list.aspx* is being requested, the category and product information can be obtained by using the following codes:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

[c#] string Age, Count;

Age =Request.QueryString[“Age”];

Count =Request. QueryString [“Count”];

### Server-side state management

In case of Server Side **State** **management**, the information will be stored on the server, the good part is that it has higher security but it can use more web server resources. The Server Side **State** **management** can again be distributed under two subheads: In Process **State** **Management** and Out of Process **State** **Management**. In Process**State** **Management** talks about Application and Session objects while Out of process **State** **Management** talks about storing **state** in SQL Server or **State** Server provided by .NET. Let us explore each one of them in detail.

#### In Process State Management Configuring In-process Mode

In-process is the default session **state** mode. To use in-process mode, set the mode attribute of the<SESSION**STATE**> element to Inproc. The following shows a sample configuration setting for in-process mode:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

<configuration>

<system.web>

<session**State** mode="Inproc" cookieless="false timeout="20"/>

</session**State**>

</system.web>

</configuration>

#### Application object

The Application object provides a mechanism for storing data that is accessible to all code running within the Web application. The information that is global to the application may be stored in application objects. For efficiency, this**state** is typically stored once and then read from many times. There are various issues, which a programmer should consider before using Application Variables. The programmer should consider issues like memory occupation, concurrency and synchronization implications of storing and accessing a global variable within a multithreaded server environment, the scalability of Application level variables as they can not be shared across a Web farm (in which an application is hosted by multiple servers) or a Web garden (in which an application is hosted by multiple processes on the same server).

In spite of these issues, well-designed application-level variables can be very powerful in Web applications. You can do a one-time (or infrequent) loading and calculation of information and then use application **state** to cache it for speedy, in-memory access during later Web requests.

#### Session object

Session object can be used for storing session-specific information that needs to be maintained between server round trips and between requests for pages. Session object is per-client basis, which means different clients generate different session objects. The ideal data to store in session-**state** variables is short-lived, sensitive data that is specific to an individual session. Each active **ASP.NET** session is identified and tracked using a 120-bit SessionID string containing URL-legal ASCII characters. SessionID values are generated using an algorithm that ensures uniqueness and SessionIDs are generated on random basis, which makes it harder to guess the session ID of an existing session. Depending on the configuration settings, the SessionIDs are communicated across client-server requests either by an HTTP cookie or a modified URL.

The following is an example of storing Session value in **ASP.NET**:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

*//to store information Session[“UserName”]=”Himanshu”;*

*//to retrieve information myname=Session[“UserName”];*

#### Storing Session State out of Process

**ASP.NET** introduces the ability to store session **state** out of process, which can be set using the session**State**element in an **ASP.NET** application's *web.config* file. The default location is in-process, as it was in classic ASP. If the mode attribute is set to **State**Server or SqlServer, however, **ASP.NET** manages the details of saving and restoring session **state** to another process or to an SQL Server database. This is appropriate because it is possible to build **ASP.NET** applications that access session **state** in the normal way, and then by switching the session**State**mode in a configuration file, that same application can be deployed safely in a Web farm environment.

#### Storing State in SQL Server Database

One of the widely used options for storing session **state** outside the server process is to keep it in an SQL Server database. **ASP.NET** supports this through the SQLServer mode in the session**State** configuration element in*web.config* file. For using this mode, you must run the *InstallSql****State****.sql* script on the database server where session**state** will be stored. This script can be found in the main Microsoft.NET directory. As soon as you run the script, it creates a table that can store client-specific **state** indexed by session ID in the *tempdb* of that SQL Server. The ASP**state** table is created in the *tempdb* database, which is not a fully logged database, thus increasing the speed of access to the data.

Following is an example of configuring *web.config* file to use SQL Server:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

<configuration>

<system.web>

<session**State** mode = "SQLServer"

"data source=192.168.1.103;user id=sa;password=" />

</session**State**>

</system.web>

</configuration>

Database enables you to store large amount of information pertaining to **state** in your Web application. Sometimes users continually query the database by using the unique ID, you can save it in the database for use across multiple request for the pages in your site.

#### The ASP.NET State Service

The **State** Service can run either on the same machine as the Web application or on a dedicated server machine. The .NET Framework provides *aspnet\_****state****.exe* that could be used as a service to maintain **state**. Using the **State**Service option is useful when you want out-of-process session **state** **management** but do not want to have to install SQL Server on the machine hosting the **state**.

Following is an example of configuring *web.config* file to use **State** Server:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/9869/Everything-you-wanted-to-know-about-State-Manageme)

<configuration>

<system.web>

<session**State** mode="**State**Server" **state**ConnectionString="10.1.1.29:23454"/>

</session**State**>

</system.web>

</configuration>