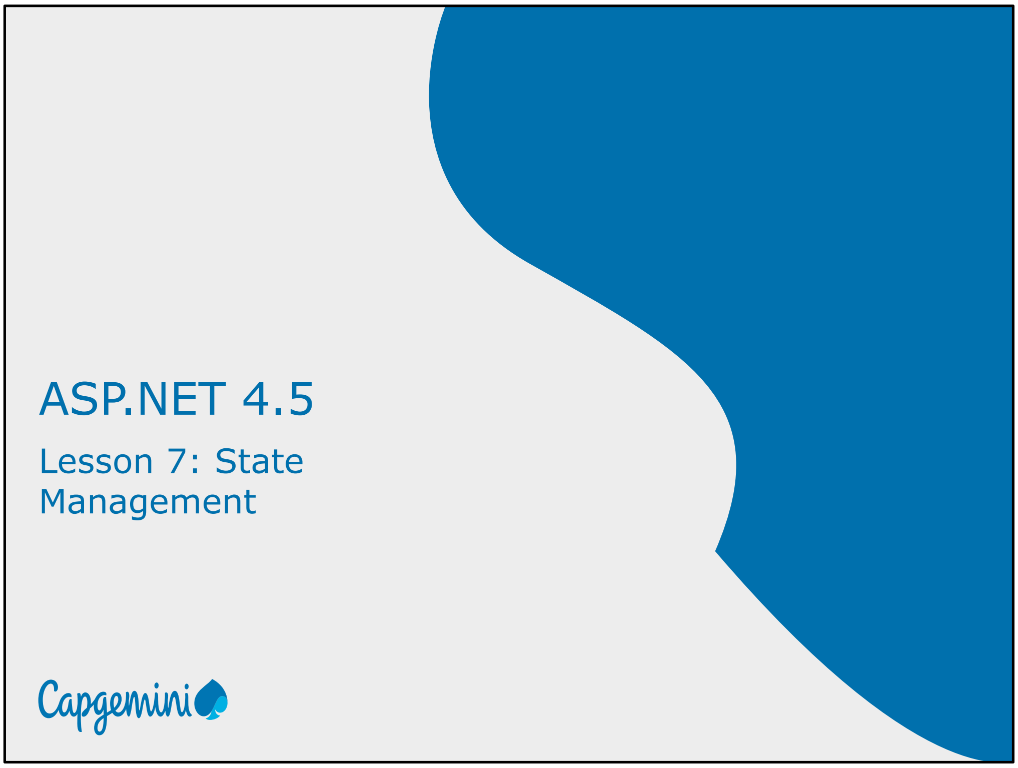
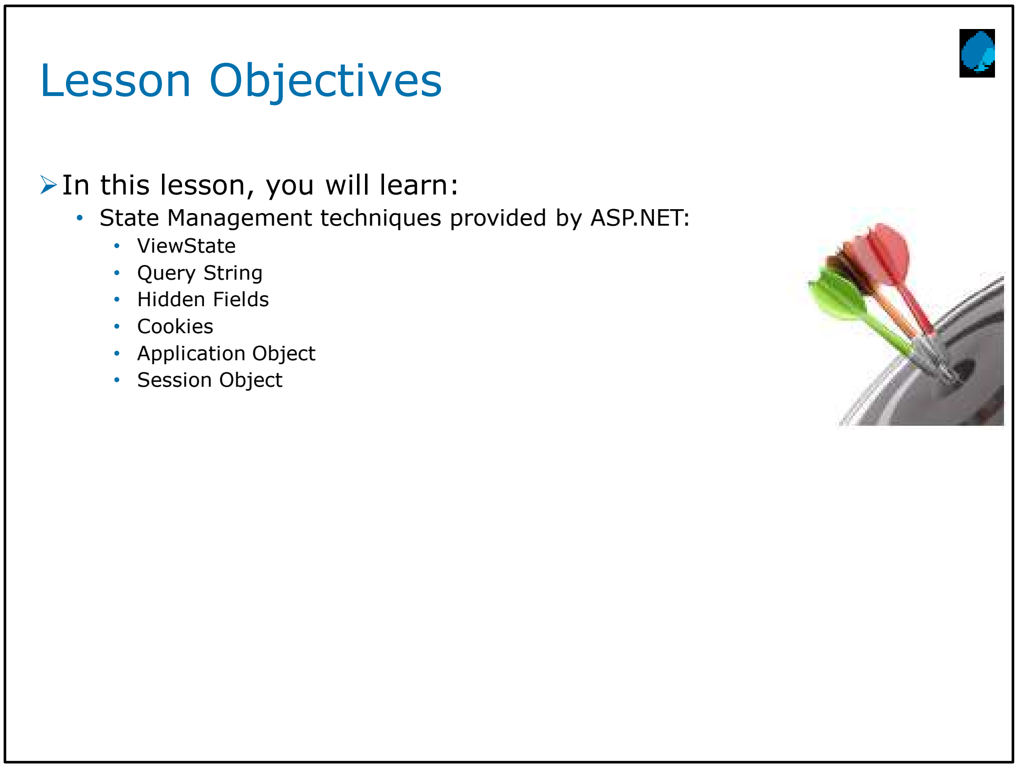


ASP.NET 4.5

Lesson 7: State Management

1



Lesson Objectives

- In this lesson, you will learn:
  - State Management techniques provided by ASP.NET:
    - ViewState
    - Query String
    - Hidden Fields
    - Cookies
    - Application Object
    - Session Object

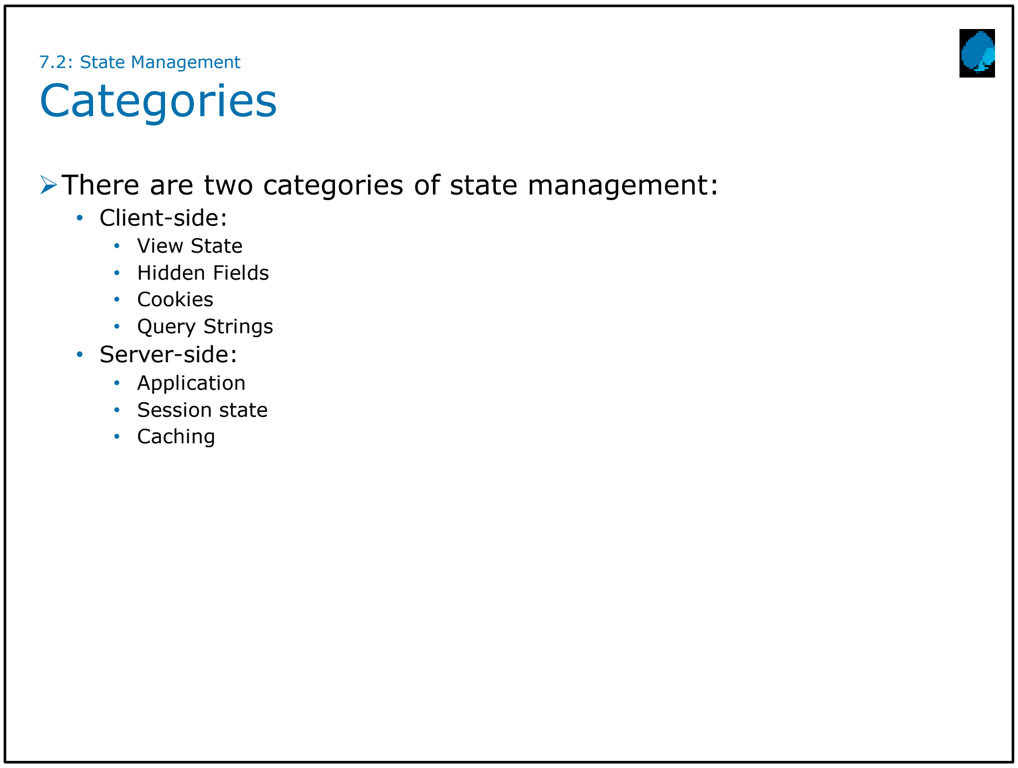
7.1: HTTP Basics

Characteristics of HTTP

- HTTP is a stateless protocol
- HTTP is also termed as a request-response oriented protocol
- The data travelling over HTTP is not persistent
- A web application has to implement various methods to maintain the persistence of data.

HTTP Basics:

- The **HTTP protocol**, the fundamental protocol of the World Wide Web, is a **stateless protocol**. It is also sometimes termed as a **request-response oriented protocol**.
- From a web server's perspective it means that every request is from a new user. The HTTP protocol does not provide you with any method of determining whether any two requests are made by the same person. However, maintaining state is important in just about any web application. A classic example is the shopping cart. If you want to associate a shopping cart with a user over multiple page requests, then you need some method of maintaining state.
- In this section, we take a look at the different options provided by ASP.NET for maintaining and managing state.



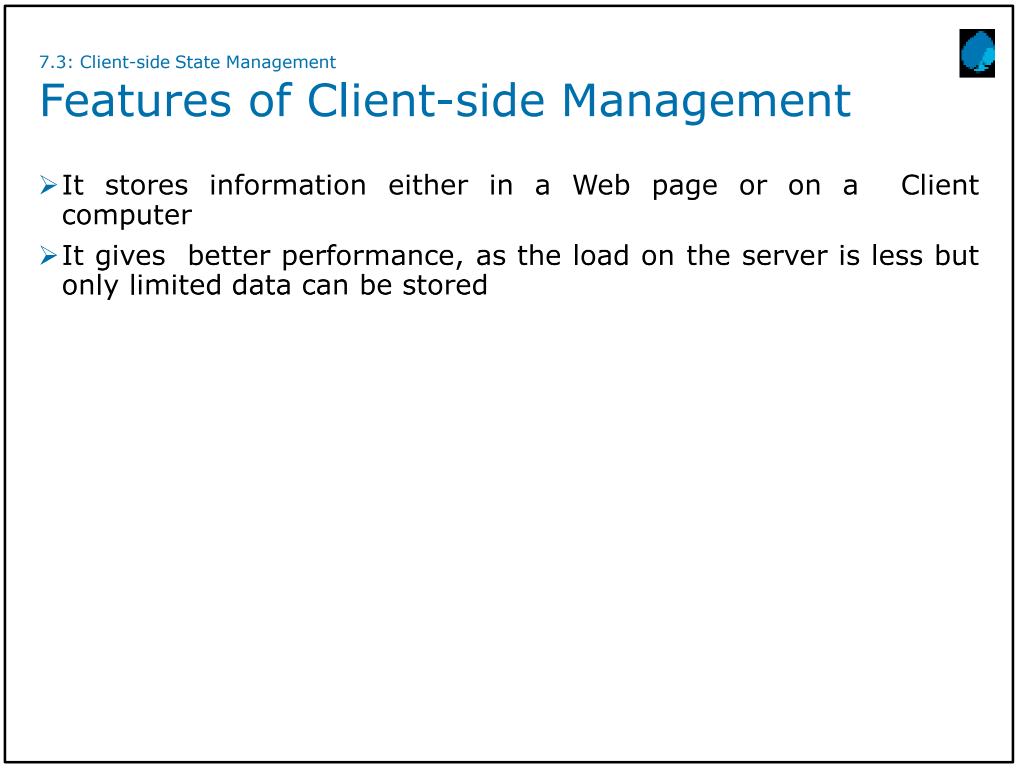
7.2: State Management

Categories

- There are two categories of state management:
  - Client-side:
    - View State
    - Hidden Fields
    - Cookies
    - Query Strings
  - Server-side:
    - Application
    - Session state
    - Caching

State Management:

- ASP.NET provides different state management options.
- State management options can be divided into two categories:
  - Client-side management
    - View State
    - Hidden Fields
    - Cookies
    - Query Strings
  - Server-side management
    - Application
    - Session state
    - Caching



7.3: Client-side State Management

Features of Client-side Management

- It stores information either in a Web page or on a Client computer
- It gives better performance, as the load on the server is less but only limited data can be stored

Client-side State Management:

- Client-side state management options involve storing information either in a **Web page** or on a **Client computer**.
- Client-side management gives a better performance, as the load on the server is less.
- Disadvantage of this option is that only limited data can be stored.

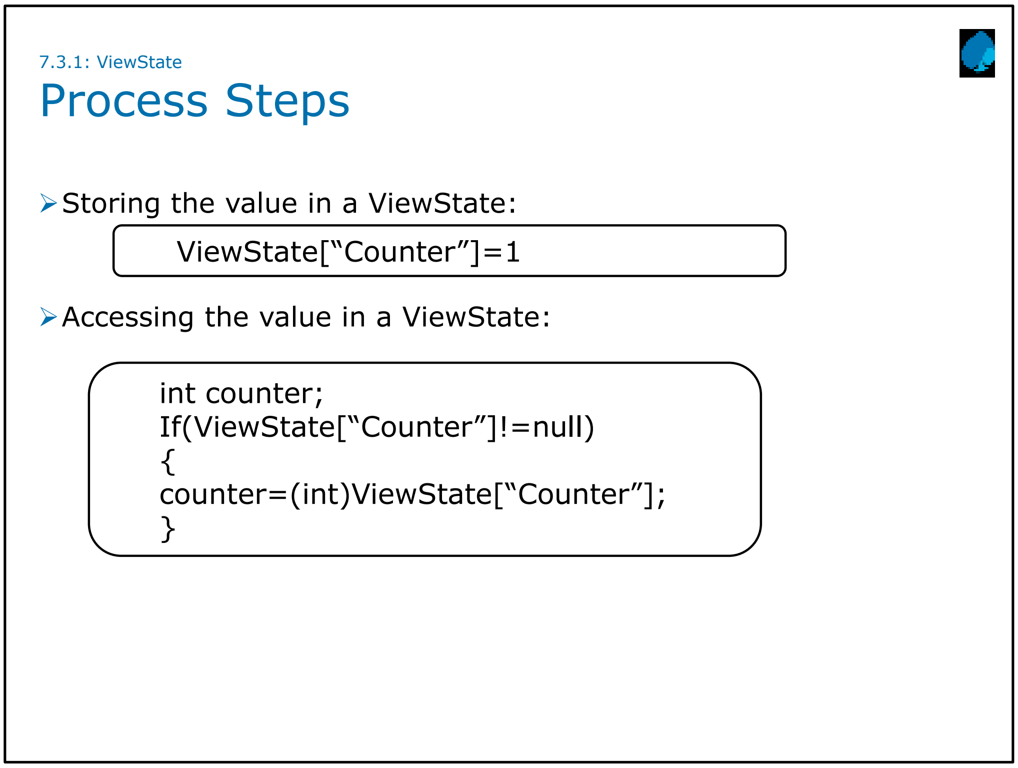
7.3.1: ViewState

Features of ViewState

- If information needs to be stored within the bounds of a single page, then go for ViewState
- ViewState enables you to retain page and control-specific values between round trips
- During Postback, ASP.NET restores the values in a hidden form field called _VIEWSTATE

ViewState:

- **ViewState** should be the first option for storing information within the bounds of a Single Page. The **ViewState property** of an ASP.NET web page enables you to retain page and control-specific values between round trips.
- Each web page and the controls on the page have the **ViewState property** that is inherited from the base Control class. The ASP.NET framework uses the **ViewState** property to automatically save the values of the Web page and each control on the Web page prior to rendering the page. During postback of a Web page to itself, ASP.NET restores the values in **_VIEWSTATE**.
- The view state is implemented with a hidden form field called **_VIEWSTATE**, which is automatically created in every Web page. When ASP.NET executes a Web page on a Web server, the values stored in the ViewState property of the page and controls on it are collected and formatted into a single encoded string. The encoded string is then assigned to the **Value** attribute of the hidden form field **_VIEWSTATE** and is sent to the client as a part of the Web page. When page is initialized during postback, ASP.NET Framework parses the ViewState string to restore the property information in the page.



7.3.1: ViewState

Process Steps

- Storing the value in a ViewState:

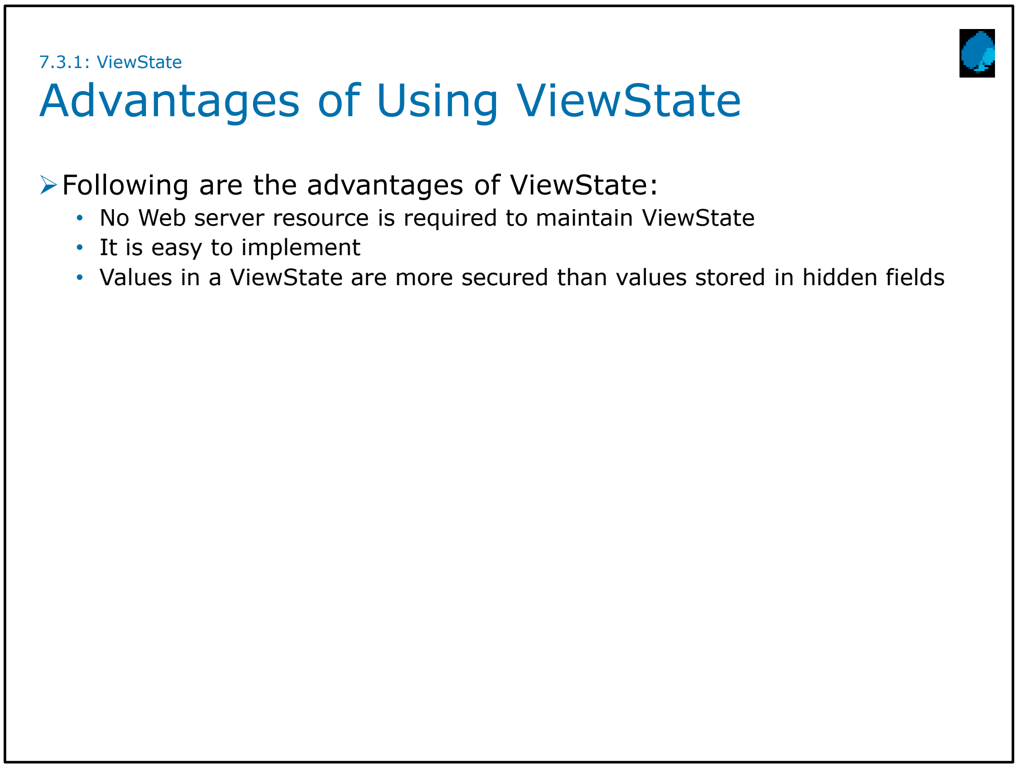
```
ViewState["Counter"]=1
```

- Accessing the value in a ViewState:

```
int counter;
If(ViewState["Counter"]!=null)
{
counter=(int)ViewState["Counter"];
}
```

<u>Storing and Accessing Values in a ViewState</u>:

- ViewState relies on a dictionary collection where each item is indexed with a unique string name. For example, the above code places 1 into ViewState Collection, and gives it a name as "***Counter***". If there is no item with the name "***Counter"***, then a new item will be added automatically. If there is already an item indexed under the name "***Counter***", then it will be replaced.
- While retrieving the value, one has to use the **Key Name**. One also needs to cast the retrieved value to the appropriate data type. The above code retrieves the counter from the ViewState and converts it to an integer.



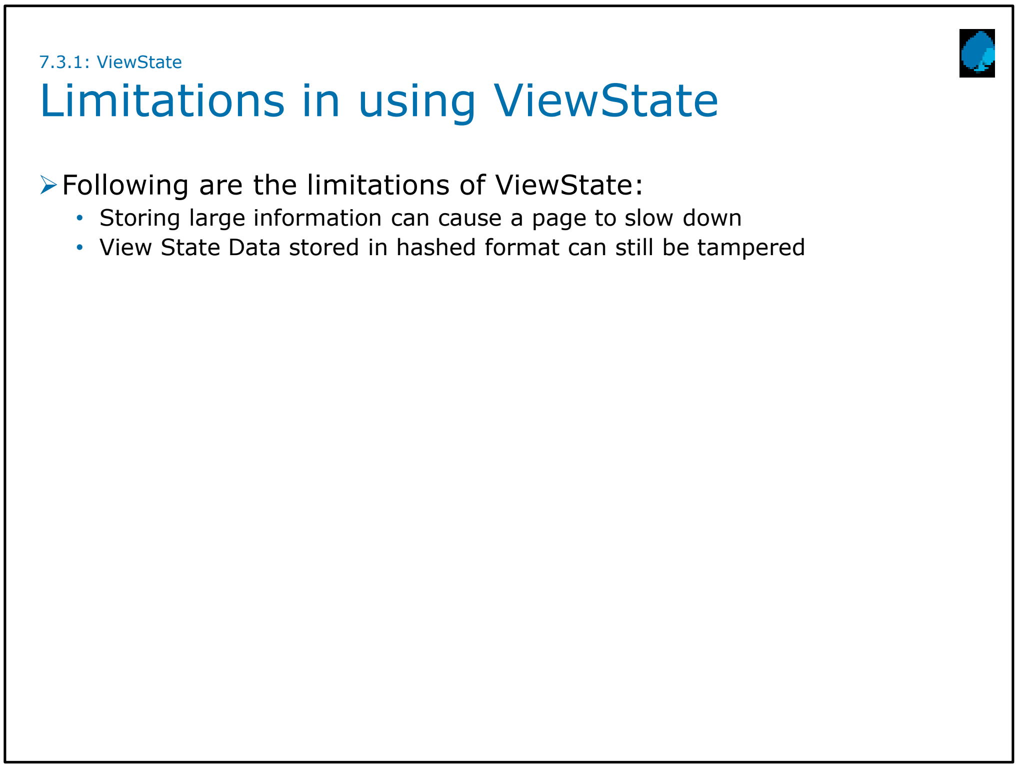
7.3.1: ViewState

Advantages of Using ViewState

- Following are the advantages of ViewState:
  - No Web server resource is required to maintain ViewState
  - It is easy to implement
  - Values in a ViewState are more secured than values stored in hidden fields

Advantages of ViewState:

- The values in a ViewState are stored in a standard **HTML format** as part of a **Web page**. Therefore no Web server resource is required to maintain it.
- The ViewState can be easily implemented in the ASP.NET Web page. You just need to set the **EnableViewState** property of a Web page and server controls.
- The values in a ViewState are hashed, compressed, and encoded for Unicode implementations. Therefore values in a ViewState are more secured than values stored in hidden fields.



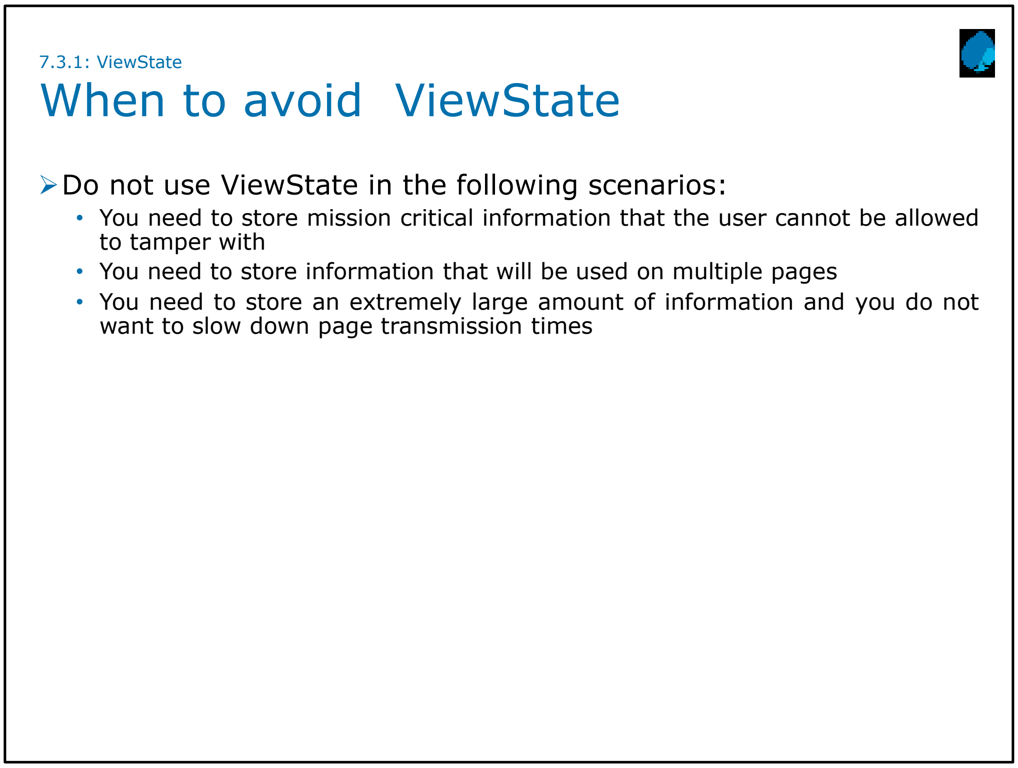
7.3.1: ViewState

Limitations in using ViewState

- Following are the limitations of ViewState:
  - Storing large information can cause a page to slow down
  - View State Data stored in hashed format can still be tampered

<u>Limitations of ViewState</u>:

- Storing large values can cause a page to slow down when user displays it or posts it. This is because ViewState is stored in the page.
- ViewState is stored in a hidden field in a page. Although ViewState stores data in a hashed format, it can be tampered with. The information in the hidden field can be seen if the page output source is viewed directly, creating a potential security issue.



7.3.1: ViewState

When to avoid ViewState

- Do not use ViewState in the following scenarios:
  - You need to store mission critical information that the user cannot be allowed to tamper with
  - You need to store information that will be used on multiple pages
  - You need to store an extremely large amount of information and you do not want to slow down page transmission times

7.3.1: ViewState

Demo: Using ViewState

- Understanding ViewState as a State Management technique

7.3.2:Hidden Fields

Concept of Hidden Fields

- One can use HTML-standard hidden fields in a Web form to store page-specific information
  - When a page is submitted to the server, the content of a hidden field is sent in the **HTTPForm** collection along with the values or other controls
  - One can use **HtmlInputHidden** control which provides the hidden field functionality

Hidden Fields:

- In ASP.NET, you can use the HTML-standard hidden fields in a Web form to store page-specific information. A hidden field does not render in a Web browser. However, you can set the properties of the hidden field. When a page is submitted to the server, the content of a hidden field is sent in the **HTTPForm** collection along with the values or other controls.
- A **hidden field** stores a single variable in its **value** property and must be explicitly added to the page. In ASP.NET, the ***HtmlInputHidden*** control provides the **hidden field** functionality.

7.3.2:Hidden Fields

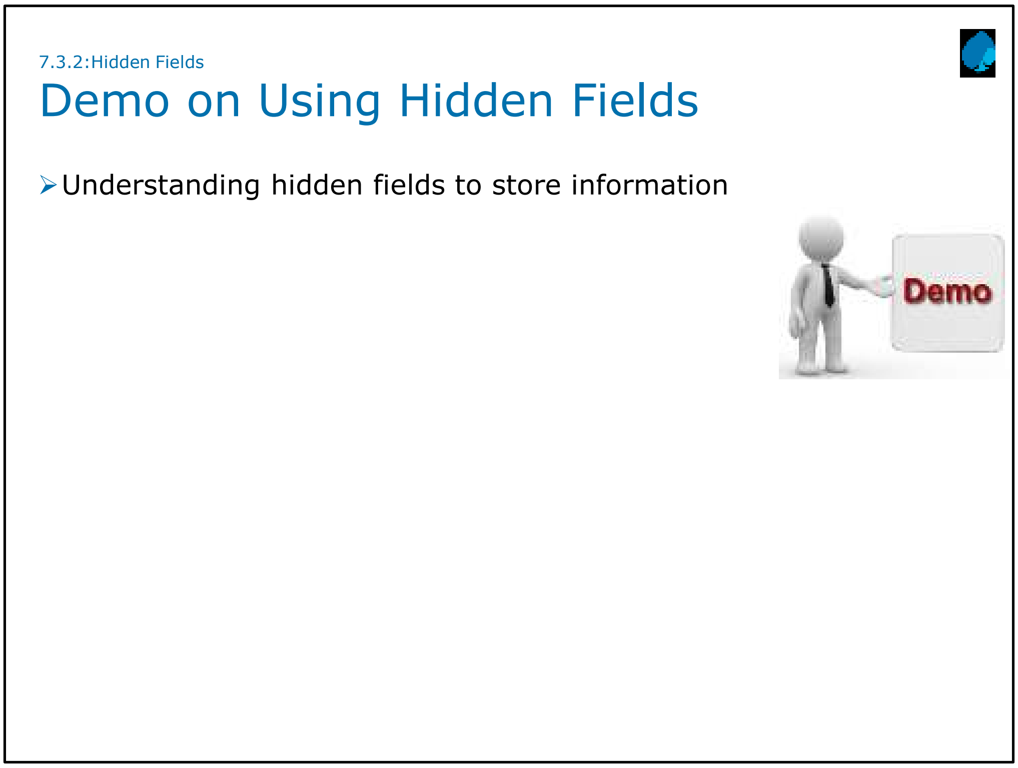
Advantages of using Hidden Form Fields

- Following are the advantages of hidden form fields:
  - A hidden field can store page-specific information without accessing any Web server or Web browser resource
  - The hidden field can be easily implemented in an ASP.NET web form page

7.3.2:Hidden Fields

Limitations of using Hidden Fields

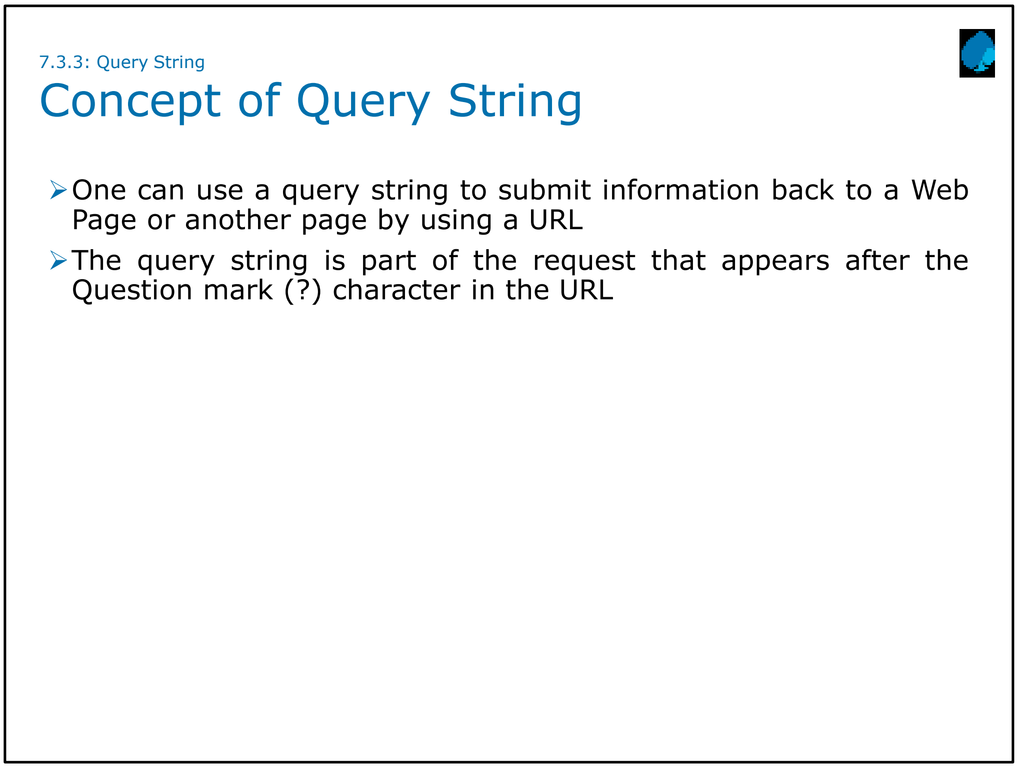
- Following are the limitations of hidden fields:
  - One can view the information stored in the hidden field by accessing the source of the Web page. Hence the values stored in hidden form fields are not very secure
  - Hidden field does not support more than a single value field to store information
  - Hidden fields are stored in a page. Hence storing large values in hidden form fields can slow down the processing of the page



7.3.2:Hidden Fields

Demo on Using Hidden Fields

- Understanding hidden fields to store information



7.3.3: Query String

Concept of Query String

- One can use a query string to submit information back to a Web Page or another page by using a URL
- The query string is part of the request that appears after the Question mark (?) character in the URL

Query String:

- A query string provides a simple way to pass information from one page to another.
- **For example:** In a logon screen, the user name can be passed to the next page in the application by using query strings.

7.3.3: Query String

Sending and Accessing the Query String

- Sending Single parameter:

```
Int recordId=10;
Response.Redirect("newpage.aspx?recordId=" +
recordId.ToString());
```

- Sending Multiple Parameters:

```
Response.Redirect("newpage.aspx?recordId=10&mode=full");
```

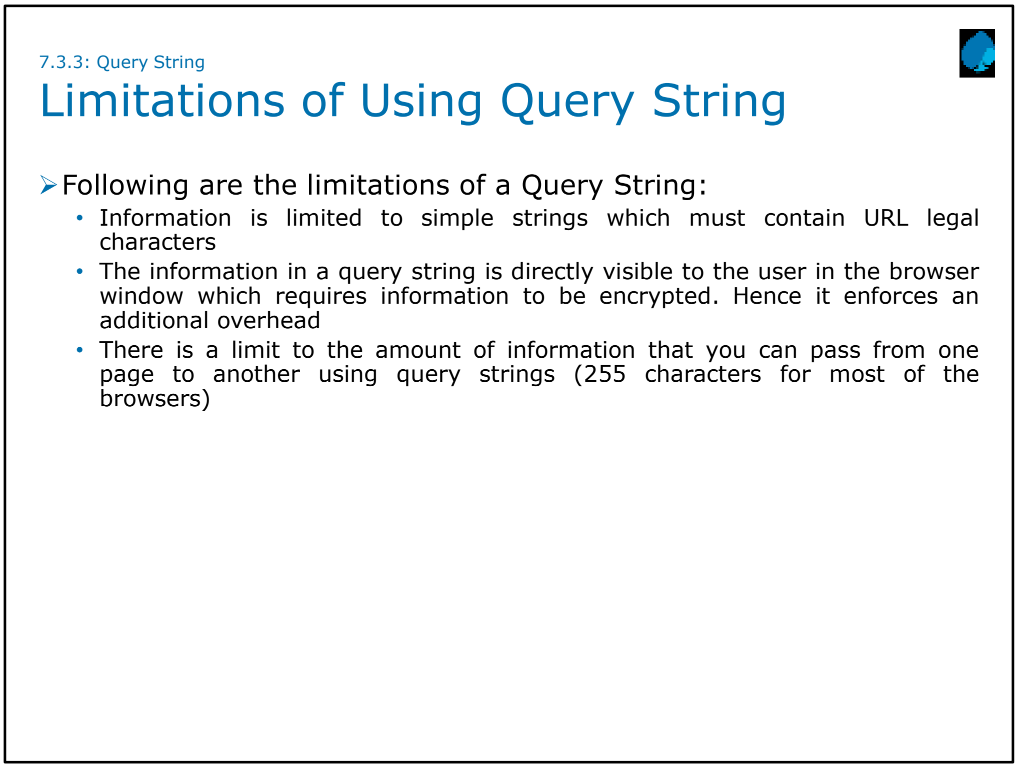
- The receiving page can receive the values using the QueryString Dictionary Collection:

```
string Id=Request.QueryString["recordId"];
```

7.3.3: Query String

Advantages of Query String

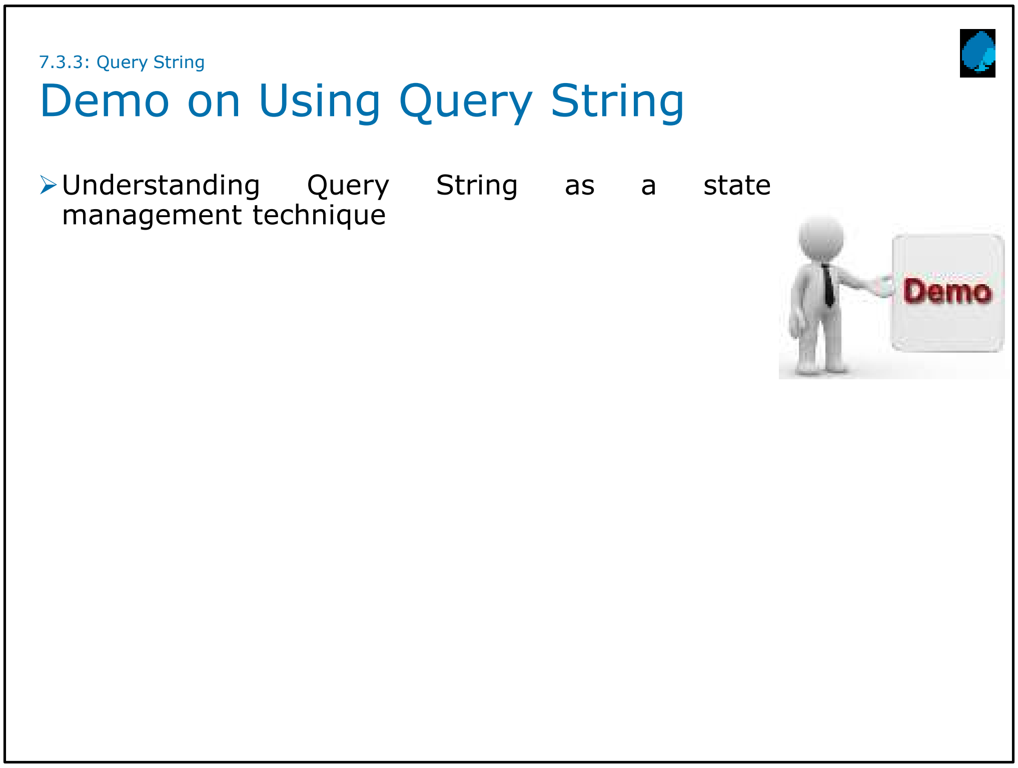
- Following are the advantages of a Query String:
  - A query string is contained in the HTTP request for a specific URL. Hence no server resource is involved to process a query string.
  - Many Web browsers support passing values in a query string.



7.3.3: Query String

Limitations of Using Query String

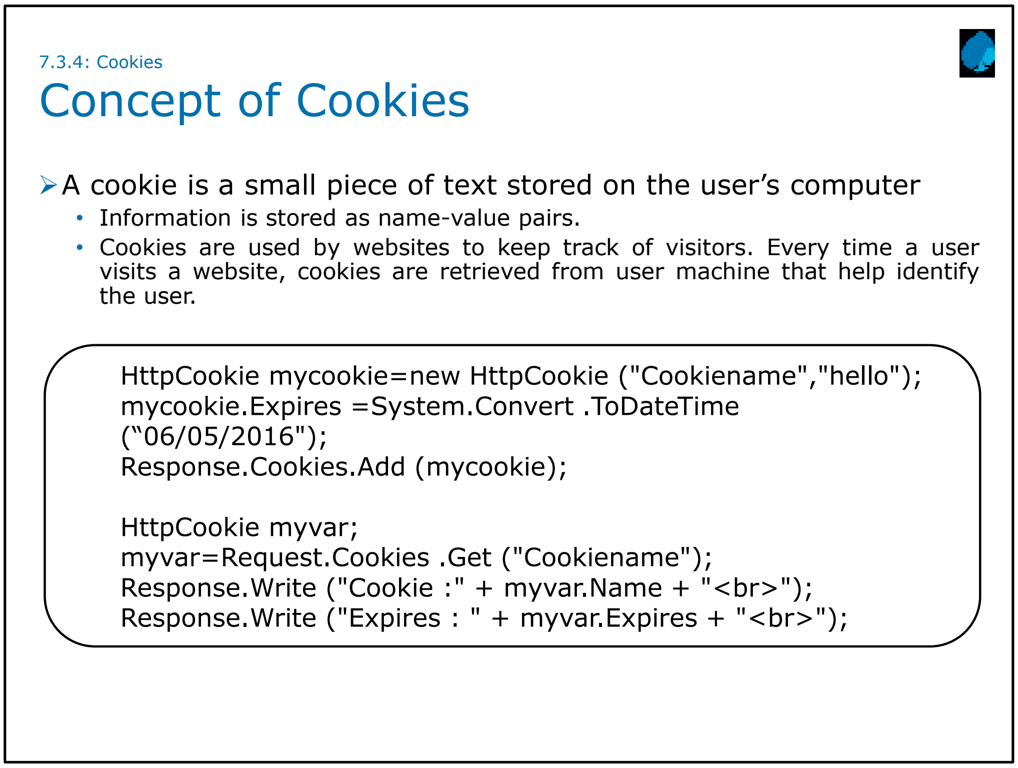
- Following are the limitations of a Query String:
  - Information is limited to simple strings which must contain URL legal characters
  - The information in a query string is directly visible to the user in the browser window which requires information to be encrypted. Hence it enforces an additional overhead
  - There is a limit to the amount of information that you can pass from one page to another using query strings (255 characters for most of the browsers)



7.3.3: Query String

Demo on Using Query String

- Understanding Query String as a state management technique



7.3.4: Cookies

Concept of Cookies

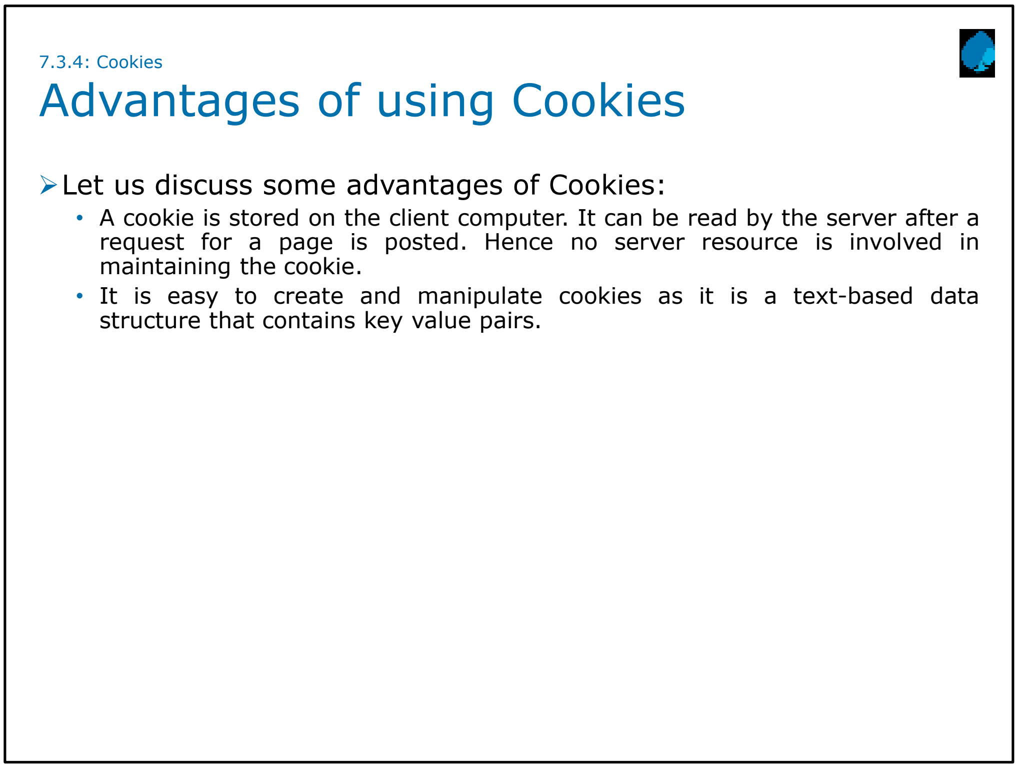
- A cookie is a small piece of text stored on the user's computer
  - Information is stored as name-value pairs.
  - Cookies are used by websites to keep track of visitors. Every time a user visits a website, cookies are retrieved from user machine that help identify the user.

```
HttpCookie mycookie=new HttpCookie ("Cookiename","hello");
mycookie.Expires =System.Convert .ToDateTime
("06/05/2016");
Response.Cookies.Add (mycookie);

HttpCookie myvar;
myvar=Request.Cookies .Get ("Cookiename");
Response.Write ("Cookie :" + myvar.Name + "<br>");
Response.Write ("Expires : " + myvar.Expires + "<br>");
```

Cookies:

- A cookie is a small amount of data that is stored either in:
  - a text file on the client file system or
  - in-memory in the client browser session.
- It contains site-specific information that the server sends to the client along with page output.
- Cookies can be **temporary** (with specific expiration times and dates) or **persistent**. A persistent cookie is saved as a text file in the file system of the client computer.
- You can use cookies to store information about a particular client, session, or application. The cookies are saved on the client device, and when the browser requests a page, the client sends the information in the cookie along with the request information. The server can read the cookie and extract its value. A typical use is to store a token (perhaps encrypted) indicating that the user has already been authenticated in your application.
- A Web browser can access a cookie from the ***HttpCookieCollection*** by using the **Request** object. If a cookie is accessed using the Request built-in object, the cookie is a read only file. The cookie is stored on the Web browser and not on the Web server. However, if you want to modify a cookie, you need to use the **Response** built in object.
- The code in the above slide shows how to create, add, and access a cookie.

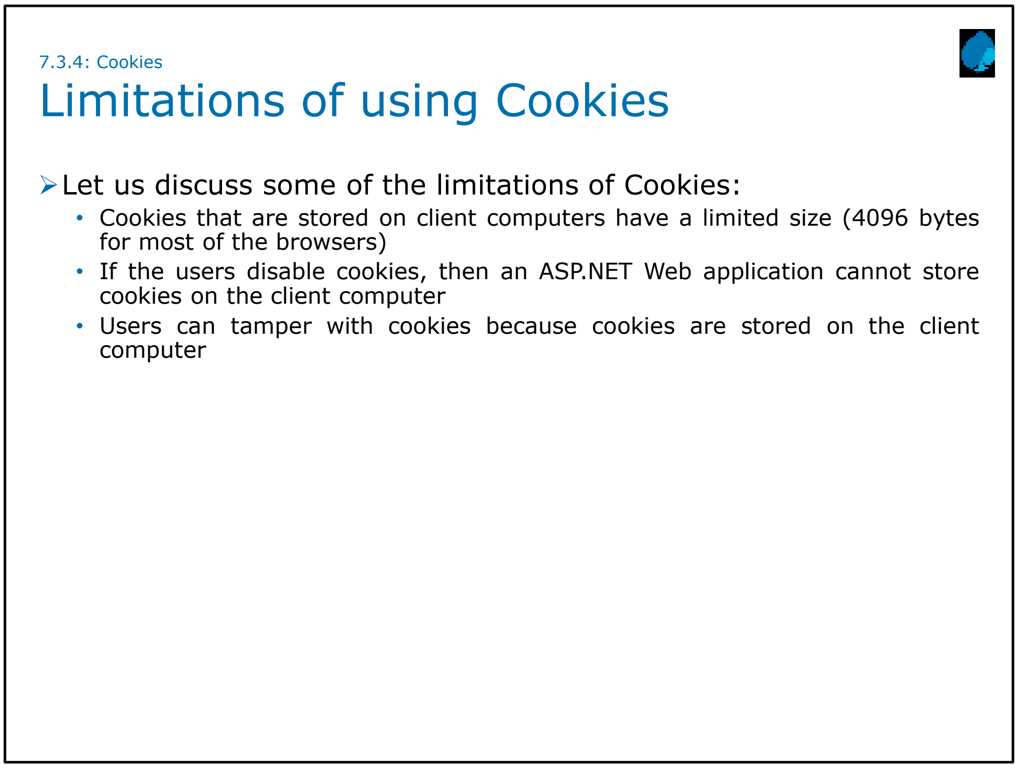


7.3.4: Cookies

Advantages of using Cookies

➢ Let us discuss some advantages of Cookies:

- A cookie is stored on the client computer. It can be read by the server after a request for a page is posted. Hence no server resource is involved in maintaining the cookie.
- It is easy to create and manipulate cookies as it is a text-based data structure that contains key value pairs.



7.3.4: Cookies

Limitations of using Cookies

- Let us discuss some of the limitations of Cookies:
  - Cookies that are stored on client computers have a limited size (4096 bytes for most of the browsers)
  - If the users disable cookies, then an ASP.NET Web application cannot store cookies on the client computer
  - Users can tamper with cookies because cookies are stored on the client computer

Demo: Using Cookies

- Understanding Cookies as a state management technique

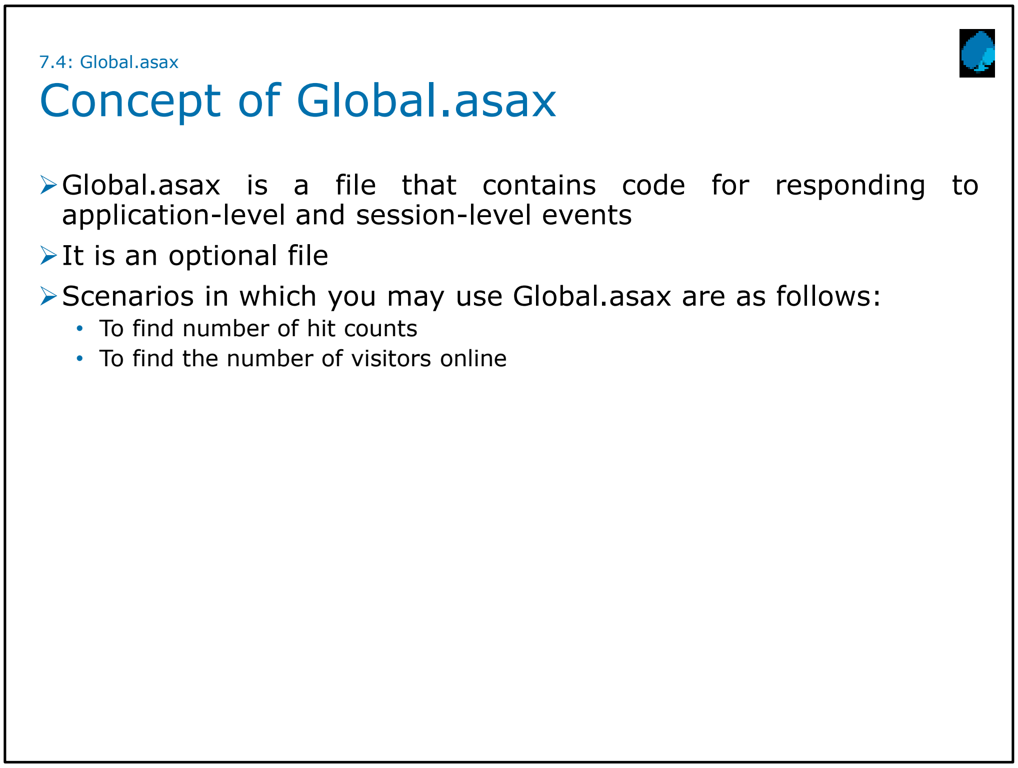
7.3.5: Control State

Control State

- Sometimes you need to store control-state data in order for a control to work properly. For example, if you have written a custom control that has different tabs that show different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips.
- The ViewState property can be used for this purpose, which can be turned off at a page level by developers, effectively breaking your control.
- To solve this, the ASP.NET page framework exposes a feature in ASP.NET called control state.
- The Control State property allows you to persist property information that is specific to a control and cannot be turned off like the ViewState property.

Control state is designed for storing a control's essential data (such as a pager control's page number) that must be available on postback to enable the control to function even when view state has been disabled. By default, the ASP.NET page framework stores control state in the page in the same hidden element in which it stores view state. Even if view state is disabled, or when state is managed using Session, control state travels to the client and back to the server in the page. On postback, ASP.NET deserializes the contents of the hidden element and loads control state into each control that is registered for control state.

Use control state only for small amounts of critical data that are essential for the control across postbacks. Do not use control state as an alternative to view state.



7.4: Global.asax

Concept of Global.asax

- Global.asax is a file that contains code for responding to application-level and session-level events
- It is an optional file
- Scenarios in which you may use Global.asax are as follows:
  - To find number of hit counts
  - To find the number of visitors online

Global.asax:

- The **Global.asax** file, is also known as the **ASP.NET application file**. It is an optional file that contains code for responding to **application-level** and **session-level events** raised by ASP.NET or by HTTP modules.
- The Global.asax file resides in the root directory of an ASP.NET application. At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the **HttpApplication** base class. ASP.NET is configured so that any direct URL request for the Global.asax file is automatically rejected. External users cannot download or view the code in it.
- The Global.asax file is optional. You create it only if you want to handle application or session events.

7.4.1: Demo: Global.asax

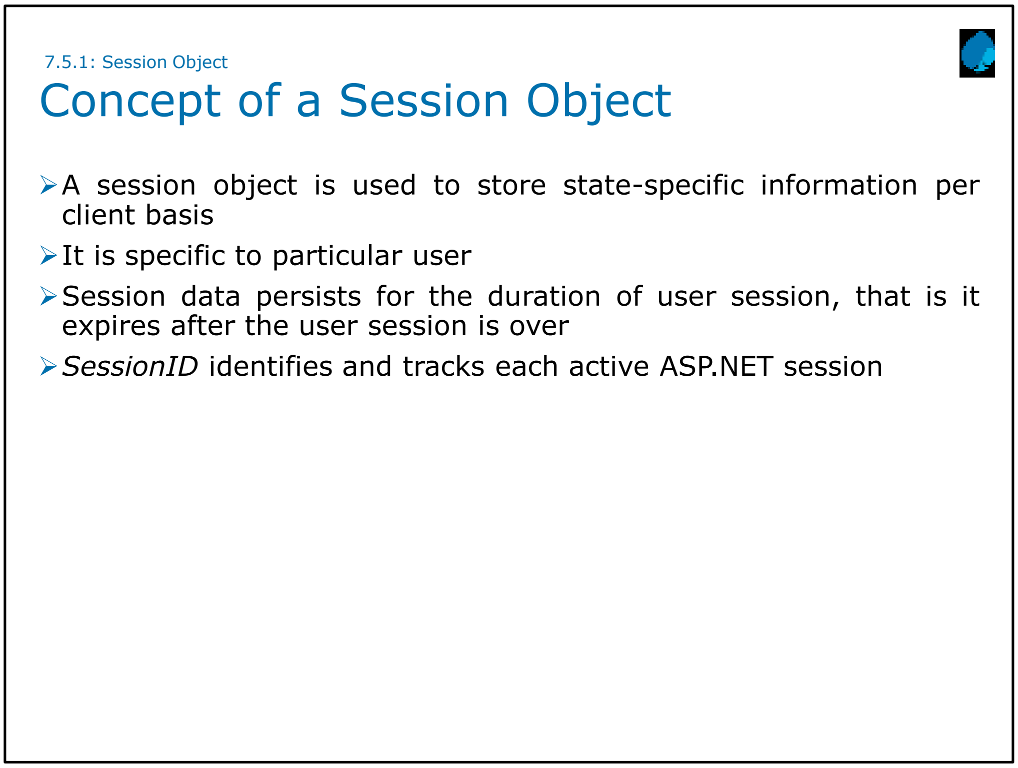
Demo

- Understanding how to use events in Global.asax file

7.5: Server-side State Management

Features of Server-side State Management

- With server side state management, one can manage application and session-related information
- Server-side options store information on the Web server



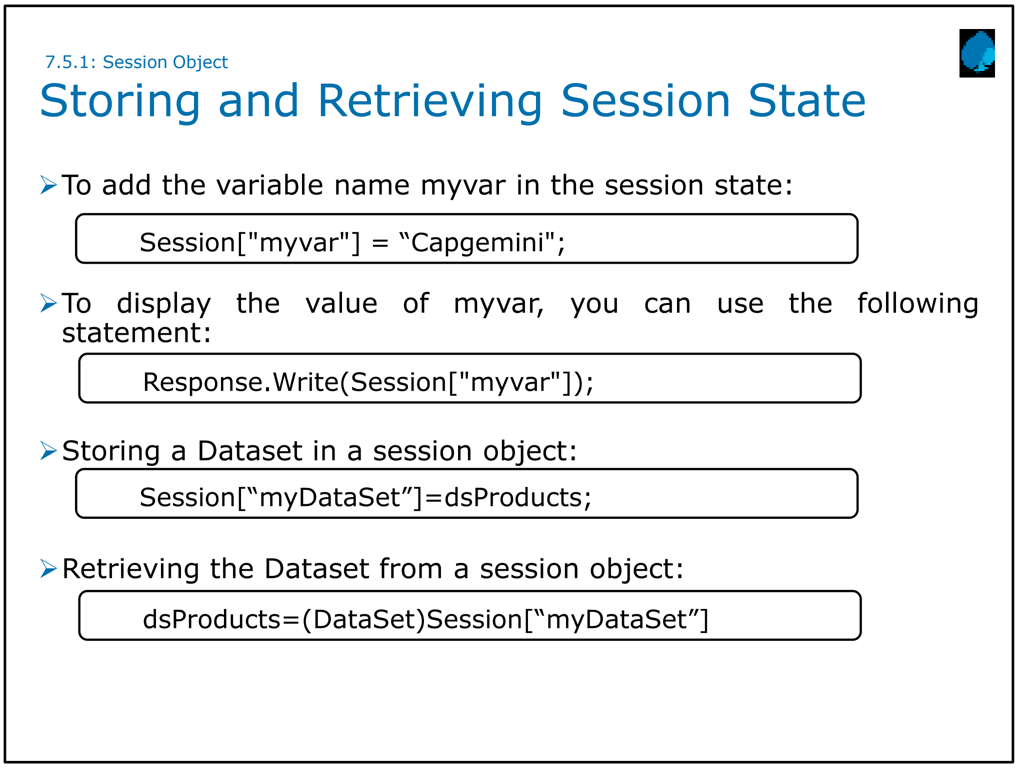
7.5.1: Session Object

Concept of a Session Object

- A session object is used to store state-specific information per client basis
- It is specific to particular user
- Session data persists for the duration of user session, that is it expires after the user session is over
- *SessionID* identifies and tracks each active ASP.NET session

<u>Session Object</u>:

- In ASP.NET, session state is used to store session-specific information for a Website. The scope of **session state** is limited to the current browser session.
- If different users are accessing a Web application, each will have a different session state. In addition, if a user exits and returns later, then the user will have a different session state.
- The session state has a built in support in ASP.NET. The built-in session state feature automatically performs the following actions:
  - Identify and classify requests coming from a browser into a logical application session on the server.
  - Store session-specific data on the server for use across multiple browser requests.
  - Raise session lifetime-related events, such as Session_OnStart and Session_OnEnd, which can be handled using application code.
- A unique 120-bit SessionID string containing ASCII characters identifies and tracks each active ASP.NET session.



7.5.1: Session Object

Storing and Retrieving Session State

- To add the variable name myvar in the session state:

```
Session["myvar"] = “Capgemini";
```

- To display the value of myvar, you can use the following statement:

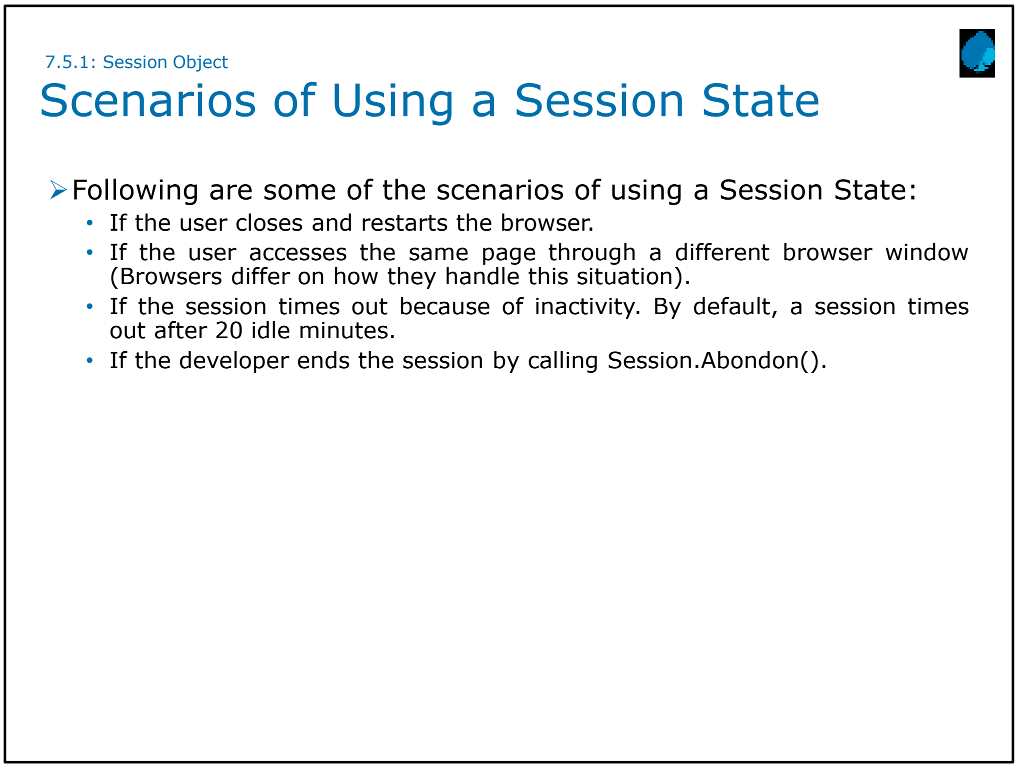
```
Response.Write(Session["myvar"]);
```

- Storing a Dataset in a session object:

```
Session[“myDataSet”]=dsProducts;
```

- Retrieving the Dataset from a session object:

```
dsProducts=(DataSet)Session[“myDataSet”]
```



7.5.1: Session Object

Scenarios of Using a Session State

- Following are some of the scenarios of using a Session State:
  - If the user closes and restarts the browser.
  - If the user accesses the same page through a different browser window (Browsers differ on how they handle this situation).
  - If the session times out because of inactivity. By default, a session times out after 20 idle minutes.
  - If the developer ends the session by calling Session.Abondon().

Using a Session State:

- Session State is global to the entire application for the current user. Session state can be lost in several ways as specified above.
- In the first two cases, the session actually remains in memory on the server. This is because the web server has no idea that the client has closed the window or changed windows.

7.5.1: Session Object

Starting and Ending a User Session

- The session is started when a user requests the first page from a Web Site
- When the first page is requested, the web server adds the ASP.NET SessionID cookie to the client computer
- To view the value of session ID:

```
Response.Write(Session.SessionID);
```

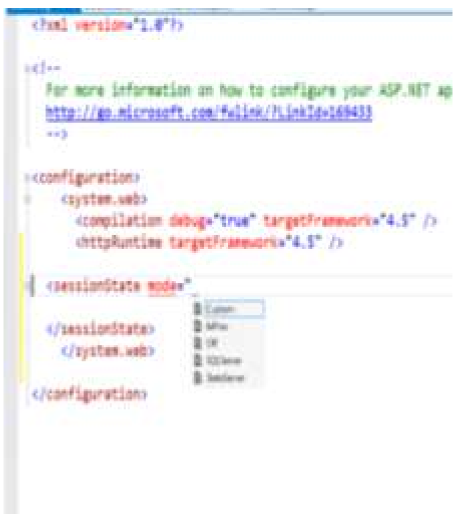
- To explicitly stop a user session:

```
Session.Abandon();
```

7.5.1: Session Object

Configuring the Session State

```
<configuration>
<system.web>
<sessionstate
mode="inproc"
cookieless="false"
timeout="20"
</system.web>
</configuration>
```



```
<?xml version="1.0"?>

<!--
  For more information on how to configure your ASP.NET ap
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->

<configuration>
    <system.web>
      <compilation debug="true" targetFramework="4.5" />
      <httpRuntime targetFramework="4.5" />

  <sessionState mode="

  </sessionState>
    </system.web>

</configuration>
```

<u>Configuring the Session State</u>:
Session state can be configured using the **<session State>** section in the application's **web.config** file as shown in the above slide.

Configuration Information:
The attributes description of the **session state** is as follows:

- **Mode:** The mode setting supports three options: inproc, sqlserver, and stateserver.
  ASP.NET supports two modes:
  - ➢ in process
  - ➢ out of process

  There are also two options for out-of-process state management:
  - ➢ memory based (stateserver)
  - ➢ SQL Server based (sqlserver)
- **Cookieless:** The cookieless option for ASP.NET is configured with this simple Boolean setting.
- **Timeout:** This option controls the length of time a session is considered valid. The session timeout is a sliding value; on each request the timeout period is set to the current time plus the timeout value. Default session timeout value is 20 minutes.
- **Sqlconnectionstring:** The sqlconnectionstring identifies the database connection string that names the database used for mode sqlserver.

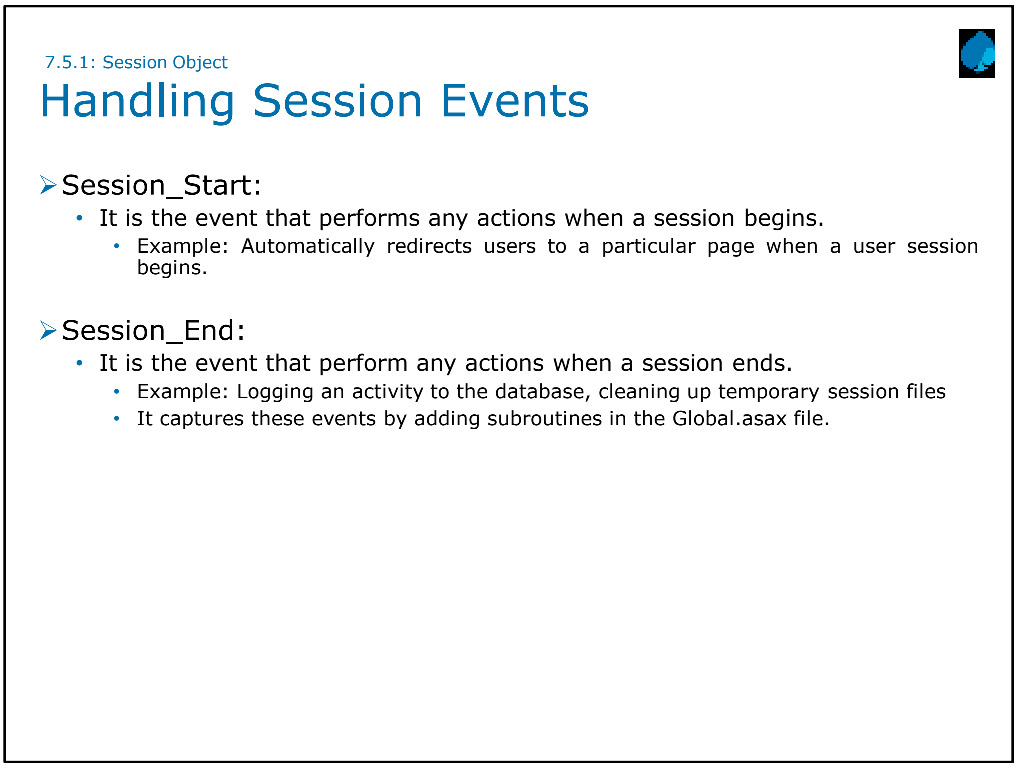
Configuring the Session State:
Configuration Information (contd.):

- **stateConnectionString : :** In the out-of-process mode stateserver, it names the server that is running the required Windows NT service: ASPState and the port setting, which accompanies the server setting

```
 <configuration>
<system.web>
<sessionstate mode=“StateServer" cookieless="false"
timeout="20"
  stateConnectionString ="tcpip:10.0.0.1:8080"
sqlConnectionString= "Data Source=10.215.60.7; user
id=sqluser; password=sqluser">
</system.web>
</configuration>
```

Disabling the SessionState:
Even though the SessionState is configured at the application-level,one can enable or disable the SessionState at the Page-level(for the pages not intended to maintain session) using the attribute EnableSessionState which takes the values True/False/Readonly

```
<%@ Page Language="C#" EnableSessionState="False"%>
```



7.5.1: Session Object

Handling Session Events

- Session_Start:
  - It is the event that performs any actions when a session begins.
    - Example: Automatically redirects users to a particular page when a user session begins.
- Session_End:
  - It is the event that perform any actions when a session ends.
    - Example: Logging an activity to the database, cleaning up temporary session files
    - It captures these events by adding subroutines in the Global.asax file.

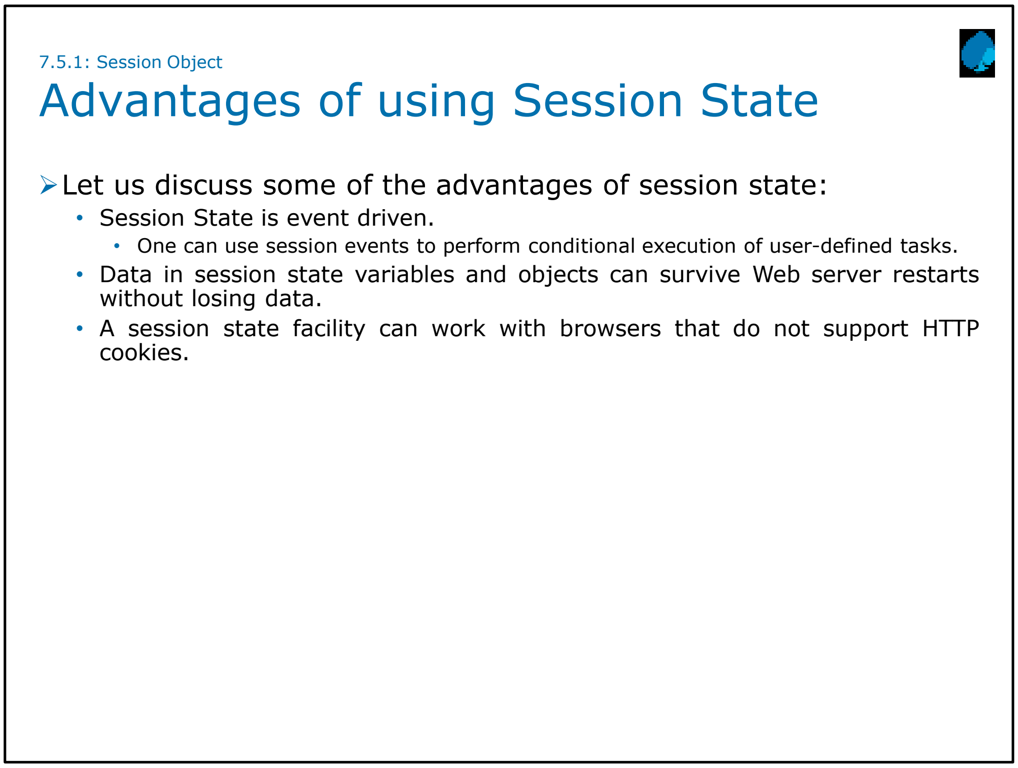
Note:
Session states have the above events which you can capture to manipulate an ASP.NET web application.

The Session Variable can be initialized in the Session_Start event

```
protected void Session_Start(object sender, EventArgs e)
       {
          Session["userName"] = null;
       }
```

One can clean up or exit the session explicitly by using the Abandon() of Session object.

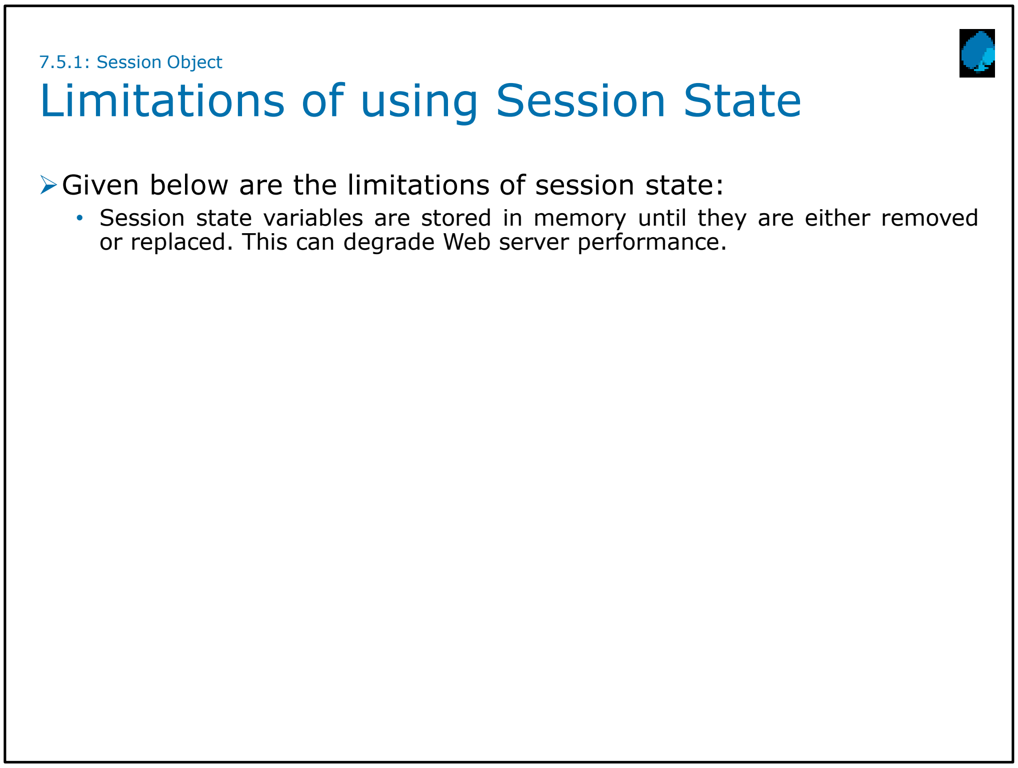
```
protected void Session_End(object sender, EventArgs e)
       {
          Session.Abandon()l;
       }
```



7.5.1: Session Object

Advantages of using Session State

- Let us discuss some of the advantages of session state:
  - Session State is event driven.
    - One can use session events to perform conditional execution of user-defined tasks.
  - Data in session state variables and objects can survive Web server restarts without losing data.
  - A session state facility can work with browsers that do not support HTTP cookies.



7.5.1: Session Object

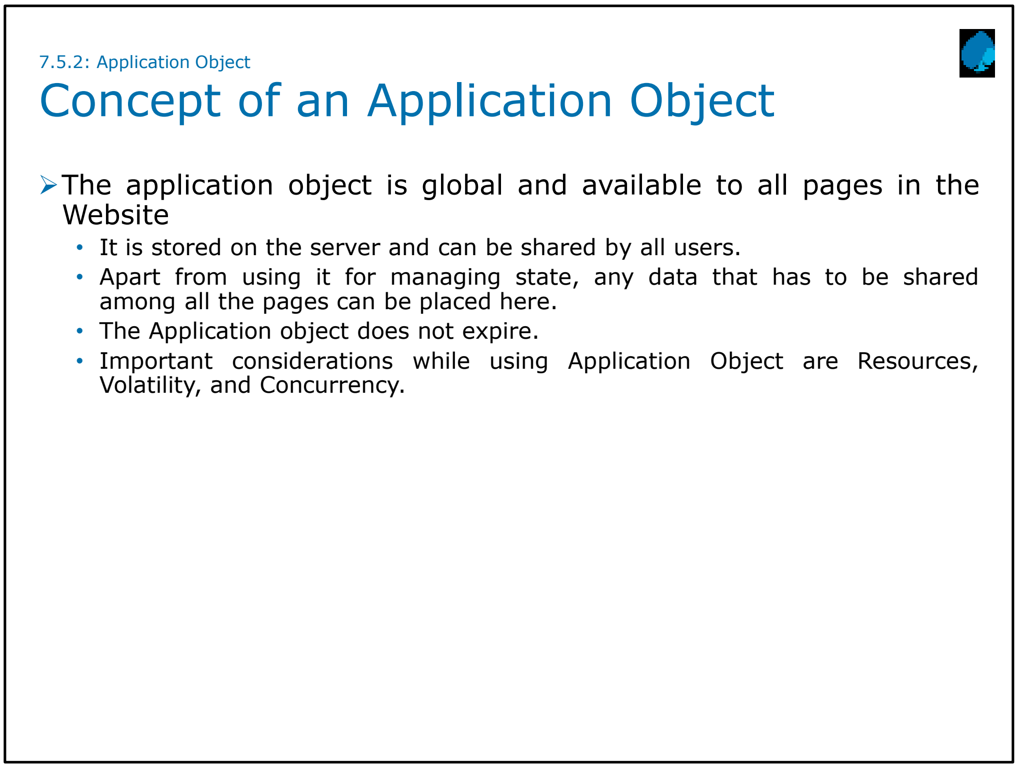
Limitations of using Session State

- Given below are the limitations of session state:
  - Session state variables are stored in memory until they are either removed or replaced. This can degrade Web server performance.

7.5.1: Session Object

Demo on Using Session State

- Understanding session State as a Server Side State Management technique



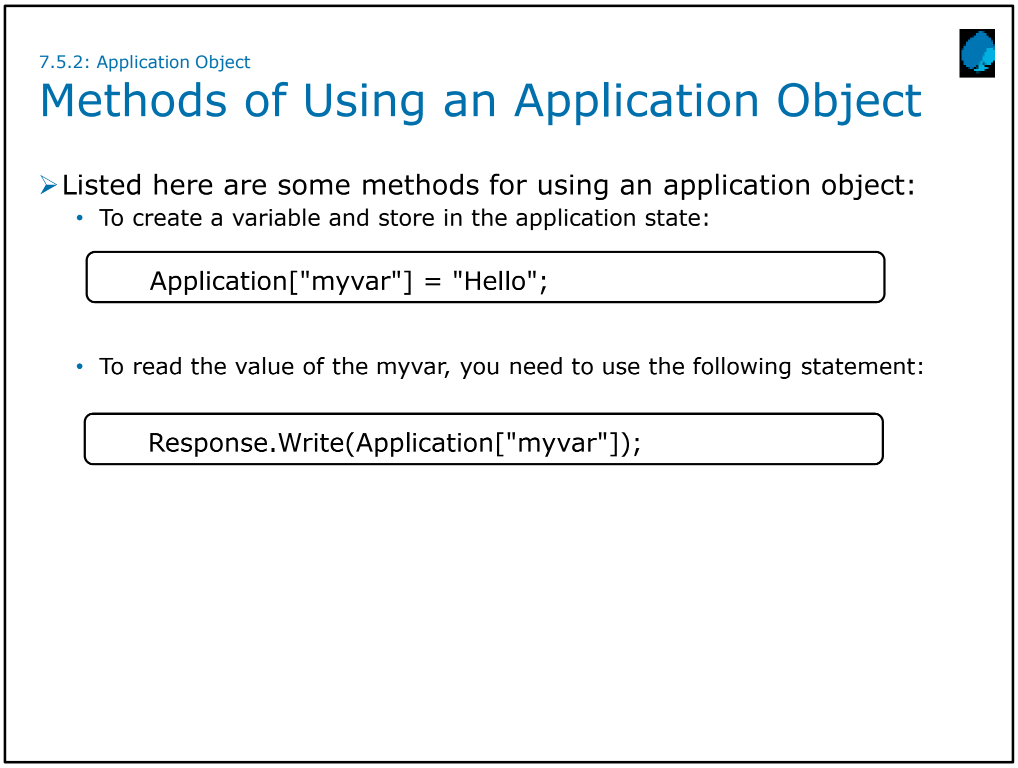
7.5.2: Application Object

Concept of an Application Object

- The application object is global and available to all pages in the Website
  - It is stored on the server and can be shared by all users.
  - Apart from using it for managing state, any data that has to be shared among all the pages can be placed here.
  - The Application object does not expire.
  - Important considerations while using Application Object are Resources, Volatility, and Concurrency.

Application Object:

- Application object is used to store data which is visible across entire application and shared across multiple user sessions. Data which needs to be persisted for “entire life of application” should be stored in application object.
- In classic ASP, application object is used to store connection strings. It is a great place to store data which changes infrequently. We should write to application variable only in **application_Onstart** event (global.asax) .
- The application object is stored on the server side and is faster than storing and retrieving information in a database. Application state is stored in an instance of the **HttpApplicationState** class.
- While using application state, you must be aware of the following important considerations state in the above slide:
  - **Resources:** Since it is stored in memory, application state is very fast compared to saving data to disk or a database. However, storing large blocks of data in application state can fill up server memory, causing the server to page memory to disk.
  - **Volatility:** The application state is stored in server memory. Hence it is lost whenever the application is stopped or restarted.
  - **Concurrency:** Application state is free-threaded. It means that application state data can be accessed simultaneously by many threads.



7.5.2: Application Object

Methods of Using an Application Object

- Listed here are some methods for using an application object:
  - To create a variable and store in the application state:

```
Application["myvar"] = "Hello";
```

  - To read the value of the myvar, you need to use the following statement:

```
Response.Write(Application["myvar"]);
```

<u>Using an Application Object</u>:

- Application state is created when each browser request is made for a specific URL. After an application state is created, the application-specific information is stored in it.
- All information stored in the application state is shared among all the pages of the Web application by using the **HttpApplicationState** class. The **HttpApplictaionState** class is accessed by using the **Application** property of the **HttpContext** object. Variables and objects added to the application state are global to an ASP.NET application.
- The application state persists until either of the following occurs:
  - the application is shut down, or
  - the **Global.asax** file is modified, or
  - the item is explicitly removed from the application state

7.5.2: Application Object

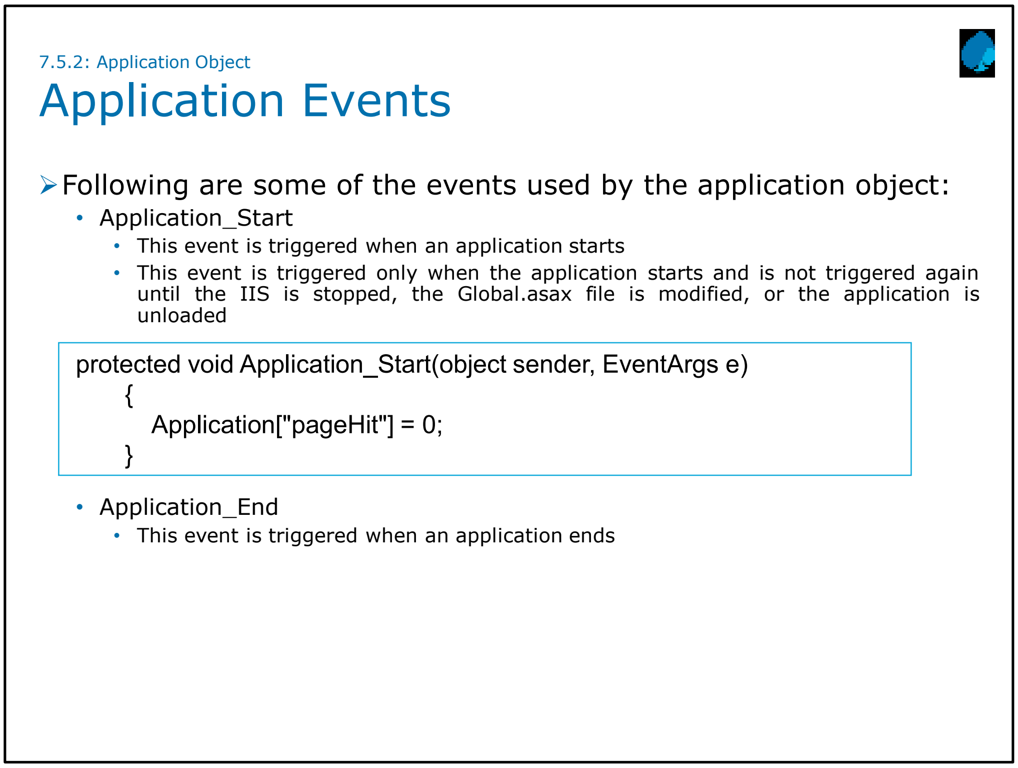
Using an Application Object

- Listed here are some methods for using an application object (contd.):
  - To remove the application variable myvar from the application state:

```
Application.Remove(["myvar"]);
```

  - To add a Dataset to an Application Object

```
DataSet ds = new DataSet();
Application["DataSet"] = ds;
```


7.5.2: Application Object

Application Events

- Following are some of the events used by the application object:
  - Application_Start
    - This event is triggered when an application starts
    - This event is triggered only when the application starts and is not triggered again until the IIS is stopped, the Global.asax file is modified, or the application is unloaded

```
protected void Application_Start(object sender, EventArgs e)
     {
        Application["pageHit"] = 0;
     }
```

  - Application_End
    - This event is triggered when an application ends

Note: Global.asax contains two events for the application state: Application_Start and Application_End.

An example to find the number of visitors to Web Application.

In the Global.asax file,using the Application_Start Event, initialize the Application Variable as given in the slide and increment the same for every Visitor visiting the page.

```
protected void Application_Start(object sender, EventArgs e)
      {
         Application.Lock();
         Application["pageHit"] = (int)Application["pageHit"];
         Application.UnLock();
      }
```

In the page_load event display the count retrieved by the Application variable

```
protected void Page_Load(object sender, EventArgs e)
      {
        int counter = (int)Application["pageHit"];
         string hitResult = String.Format("Visitor # : {0} ", counter);
         lblPageHit.Text = hitResult;
      }
```

7.5.2: Application Object

Significant Issues

- Here are some issues encountered in using application state:
  - The memory occupied by variables stored in an application state is not released until the value is either removed or replaced. This is an overhead on the Web server and the server response will be slow
  - Multiple pages within an application can access values stored in an application state simultaneously. It requires explicit synchronization methods to avoid deadlocks and access violations.

Note:
Since application state variables are global to an application, it is important to consider the above issues while storing any value in an application state variable.

7.5.2: Application Object

Advantages of Using Application State

- Following are some of the advantages of using an application state:
  - Application state is easy to use and is consistent with other .NET Framework classes
  - Storing information in application state involves maintaining only a single copy of information

7.5.2: Application Object

Limitations

- Let us see some of the limitations of using an application state:
  - The data stored in an application state is lost when the Web server containing the application state fails due to a server crash, upgrade, or shutdown.
  - Application state requires server memory and can affect the performance of the server and the scalability of the Web application.

7.5.2: Application Object

Demo on Using Application Object

- Understanding Application Object as a State Management technique

7.6 HTML 5 -Web Storage

HTML 5 -Web Storage

- One alternative to cookies in scenarios where you would like to cache data on the client is to use web storage (sometimes known as DOM storage).
- Browsers starting from IE 8, Firefox 4.5, Safari 4, Chrome 4, and Opera 10.50 all support web storage.
- The benefit of using web storage over cookies is that the browser doesn't have to round-trip the data from the client to the server with every request.

You have 5MB (Firefox, Chrome, and Opera) or 10MB (IE) of space. Although you can only store strings, you can encode and store binary or more complex objects using JSON.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
   <script>
      function saveDraft() {
window.localStorage.uname = document.getElementById("txtUName").value;
window.localStorage.email =  document.getElementById("txtEmail").value;
window.localStorage.dob =   document.getElementById("txtDob"). value;
window.localStorage.age =   document.getElementById("txtAge"). value;
window.localStorage.phone = document.getElementById("txtPhone"). value;
window.localStorage.blog = document.getElementById("txtBlog"). value;
      }
 function loadDraft() {
 document.getElementById("txtUName").value = window.  localStorage.uname;
 document.getElementById("txtEmail").value = window.localStorage. email;
 document.getElementById("txtDob").value = window.localStorage. dob;
 document.getElementById("txtAge").value = window.localStorage. age;
 document.getElementById("txtPhone").value = window.localStorage.phone;
 document.getElementById("txtBlog").value = window.localStorage. blog;
      }
    </script>
</head>
```

```
<body>
   <form id="form1" runat="server">
   <div>
      <table style="width:59%;">
<tr><td class="auto-style1">Username</td>
<td>
<asp:TextBox ID="txtUName" runat="server"></asp:TextBox>
</td>
</tr>
 <tr>
<td class="auto-style1">Email</td>
<td>
<asp:TextBox ID="txtEmail" runat="server" TextMode="Email"></asp:TextBox>
</td>
</tr>
 <tr>
<td class="auto-style1">Date of Birth</td>
<td>
<asp:TextBox ID="txtDob" runat="server" TextMode="Date"></asp:TextBox>
</td>
</tr>
 <tr>
<td class="auto-style1">Age In Years</td>
<td>
<asp:TextBox ID="txtAge" runat="server" TextMode="Number"></asp:TextBox>
</td>
</tr>
 <tr>
<td class="auto-style1">Phone</td>
<td>
<asp:TextBox ID="txtPhone" runat="server" TextMode="Phone"></asp:TextBox>
</td>
</tr>
      <tr>
<td class="auto-style1">Blog address</td>
<td>
<asp:TextBox ID="txtBlog" runat="server" TextMode="Url"></asp:TextBox>
</td>
</tr>
      <tr>
<td>
<asp:Button ID="btnReset" OnClick="btnReset_Click"runat="server" Text="Reset"
/> </td>
<td class="auto-style1">
<button id="load" onclick="loadDraft()">Load Draft</button>
</td>
<td class="auto-style1">
<button id="draft" onclick="saveDraft()">Save Draft</button>
<input type="submit"/></td>
</table>
   </div>
   </form>
</body>
</html>
```

7.6 HTML 5 -Web Storage

Demo

- HTML 5 Web Storage in ASP.Net Web Application

7.7:State Management Best Practices and Guidelines

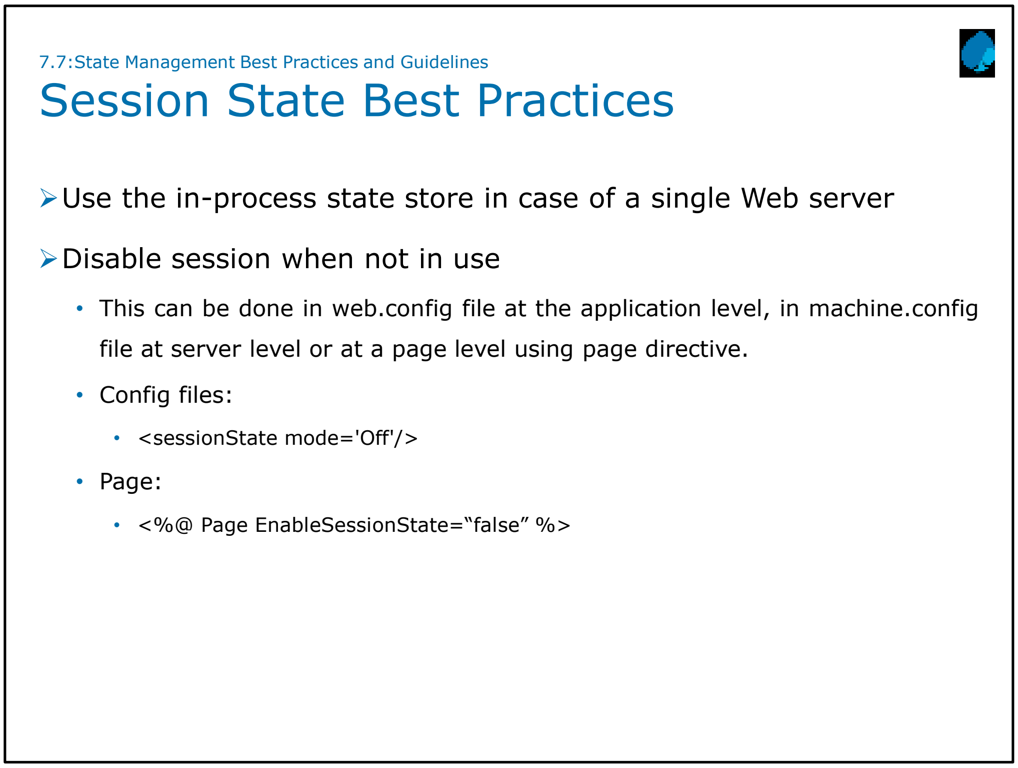
State Management Best Practices

- Use cookies, query strings, and hidden controls for storing lightweight, user-specific state that is not sensitive
- Do not use cookies, query strings, and hidden controls to store security-sensitive information. This is because the information can be easily read or manipulated
- While serializing state, prefer simple/basic types such as Int, Byte, Decimal, String rather than complex objects, to reduce the impact of serialization

7.7:State Management Best Practices and Guidelines

Application State Best Practices

- Store only read-only data in application state to avoid server affinity



7.7:State Management Best Practices and Guidelines

Session State Best Practices

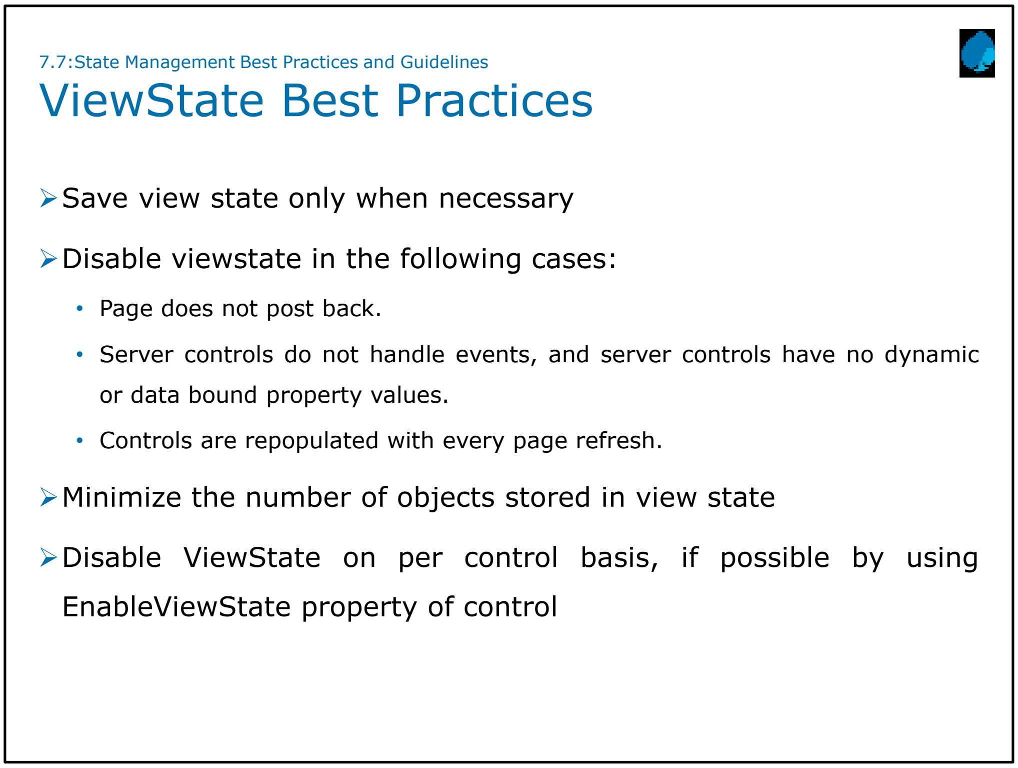
- Use the in-process state store in case of a single Web server
- Disable session when not in use
  - This can be done in web.config file at the application level, in machine.config file at server level or at a page level using page directive.
  - Config files:
    - <sessionState mode='Off'/>
  - Page:
    - <%@ Page EnableSessionState="false" %>

7.7:State Management Best Practices and Guidelines

Session State Best Practices (Contd…)

- Minimize the amount and complexity of data stored in a session state. The larger and more complex the data is, the cost of serializing/deserializing of the data is higher
- Do not use too many session variables
- Ensure that session timeout is reasonable
- If the session state only needs to be read, specify the readonly flag for session state, to avoid the write-back

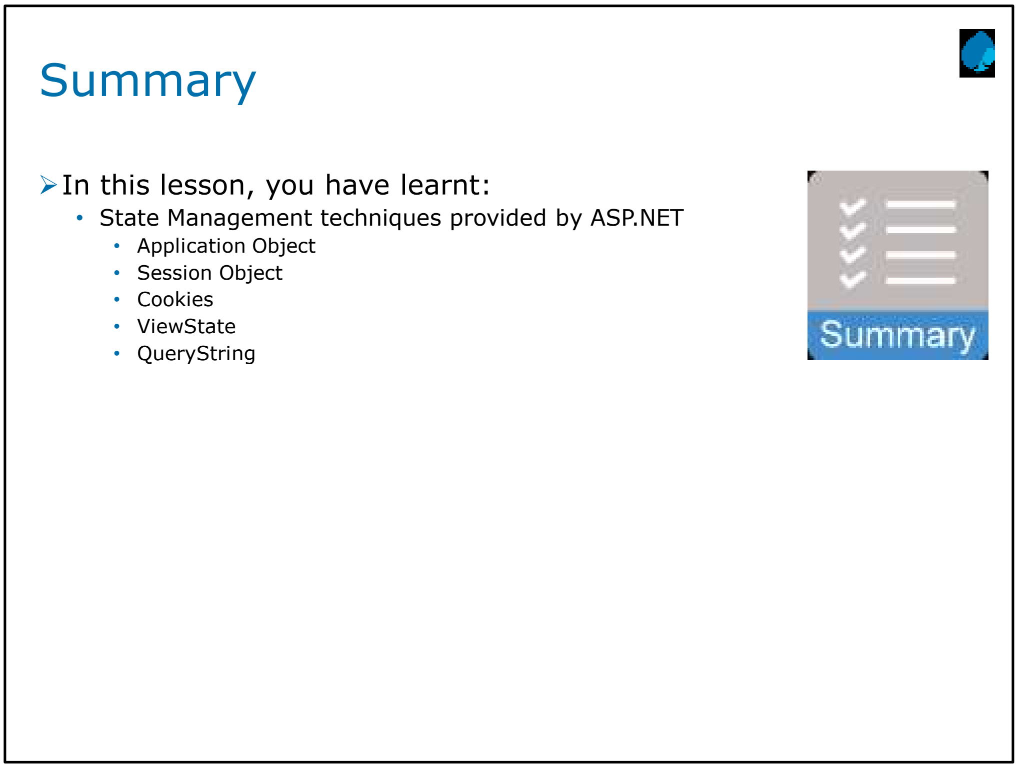
```
<%@ Page EnableSessionState="readonly" %>
```



7.7:State Management Best Practices and Guidelines

ViewState Best Practices

- Save view state only when necessary
- Disable viewstate in the following cases:
  - Page does not post back.
  - Server controls do not handle events, and server controls have no dynamic or data bound property values.
  - Controls are repopulated with every page refresh.
- Minimize the number of objects stored in view state
- Disable ViewState on per control basis, if possible by using EnableViewState property of control



Summary

- In this lesson, you have learnt:
  - State Management techniques provided by ASP.NET
    - Application Object
    - Session Object
    - Cookies
    - ViewState
    - QueryString

Review Question

- Question 1: Which object in ASP.NET can be used to store data that is global:
  - Session
  - Static
  - System
  - Application
- Question 2: What data is stored at the server's memory (select all that is appropriate)?
  - Session
  - Application
  - Cookie
  - ViewState
  - Hidden Variable