

Maintaining State on your Web Site

Introduction

- Within the client-server model, when a client wants to view a particular HTML or ASP.NET page he first requests the page and then once the page is found it is returned to him.
- At this point the conversation between the client and the server is over.
- **The web server does not keep information on what clients it has spoken to recently**, or what requests were made.
- For this reason Web sites are often referred to as **stateless**.
 - An application is said to have state if it persists information for each user.

Introduction

- Variables within a Web Form Page do not retain their values between requests.
 - Requesting a Web Form Page is similar to starting and stopping an application. After the page is finished being processed, you must start all over again with a blank slate.
- If you need to maintain the value of a variable across multiple page requests, you need to do something special.
- There are four methods of maintaining the state of variables within your web site. These are:
 - **View State** to retain the state of variables for a particular page
 - **Cookies** to retain information from page to page
 - **Session State** to retain the state of variables for a particular user
 - **Application State** to retain the state of variables for all users

Using View State

- ViewState enables you to **preserve the state of a page when posting the page back to itself.**
- It's important to understand that View State does not work when posting from one page to another.
 - You can only take advantage of View State when performing postbacks.
- Every Web Form control automatically preserves the values of all of its properties through View State.
 - For example, if you type an item in a textbox and perform a postback the value in the textbox will be retained.
- View State also applies to the items displayed in a GridView or DetailsView.
 - A GridView or DetailsView automatically retains the values of all of its rows in View State.

Using View State

- View State works by taking advantage of a **hidden form field** that is automatically added to every Web Form Page.
 - You can see this hidden form field if you select View Source when displaying a Web Form Page in a browser.
- Immediately before a page is rendered, the values of the properties of all controls in a page are converted into a string.
- When the page is rendered, this string is assigned to the hidden `_VIEWSTATE` form field.
- When the page is posted back to the server, the string is parsed and all the values are reassigned to the properties of the controls in the page.

Using View State

Disabling View State

- View State can be both a blessing and a curse.
- View State enables you to magically retain the state of all the controls on a page.
- However, the hidden VIEWSTATE form field **can become very large**, which can slow down the rendering of a page.
- You don't always need to take advantage of View State.
 - If you don't need to preserve the state of controls across post backs, you don't need to use View State.
- You can **disable View State** for any control by assigning the value False to its EnableViewState property.
 - You should always disable View State for controls such as the Repeater, DataList, and DataGrid controls when you are simply displaying data.
 - You also can disable View State for all the controls in a page by using the DOCUMENT EnableViewState property. To disable View State for a page, select DOCUMENT in the Properties window and assign the value False to the EnableViewState property.

Using View State

Adding Custom Information to View State

- You can **add your own data** to View State by taking advantage of the ViewState collection.
- Any data that you add to the ViewState collection will be added to the `_VIEWSTATE` hidden form field and will be preserved across postbacks.
 - For example, the following statement adds the string "Hello World" to the ViewState collection:

```
ViewState["myItem"] = "Hello World";
```

Using View State

Adding Custom Information to View State

- After an item has been added to the ViewState collection, you can retrieve the item by passing the name of the item to the collection:

```
Response.Write(ViewState['myItem']);
```

- You can add almost any object to the ViewState collection.
 - Including strings, integers, ArrayLists, and even DataSets.
 - The only requirement is that the object must be serializable.
- However, remember that **anything you add to View State must be added to the _VIEWSTATE hidden form field.**
 - Consequently, adding a DataSet to View State can significantly impact the rendering time for a page.

Using View State

Protecting View State

- By default, View State is **not encrypted**.
 - In theory, anyone requesting a page can copy the page to their local hard drive, modify the value of the hidden `_VIEWSTATE` form field, and submit the modified page back to the server.
 - For example, if you are storing a shopping cart in View State, someone could modify the price of all the products in the shopping cart and submit the modified page.
- You can **protect the integrity of View State** by adding a Message Authentication Code (MAC) to the hidden `_VIEWSTATE` form field.
 - After the View State MAC is enabled, an exception is thrown if someone tampers with the View State.
 - You can enable the View State MAC for a particular Web Form Page by selecting DOCUMENT in the Properties window and assigning the value True to the `EnableViewStateMAC` property.

Not any more...

<http://blogs.msdn.com/b/webdev/archive/2014/09/09/farewell-enableviewstatemac.aspx>

Using View State

Protecting View State

- You can enable the View State MAC for an application or directory by adding the following tag to the System.Web section of the Web.Config file:

```
<pages enableViewStateMac='true' />
```

- The View State MAC is disabled by default. Enabling the View State MAC adds some extra work to the processing of the page, so it has an impact on performance. If you don't need it, don't enable it.
- Enabling the View State MAC **does not result in the View State being encrypted.**
- If you want to encrypt the contents of View State, you need to enable the View State MAC and perform an additional step.
 - You need to add the following tag to the System.Web section of the Web.Config file:

```
<machinekey validation="3DES" />
```
 - This tag causes all View State to be **encrypted using the Triple Data Encryption Standard**. Again, this has an impact on performance, so don't enable it unless you need it.

Using Cookies

- A web site does not have to be stateless.
- It can use **cookies** to persist information over lengthy periods of time.
- Cookies are small bits of information, such as strings and numeric values, stored on a clients computer for a **specified period of time**.
 - When cookies are created on the client's computer, the developer needs to specify when they expire.
 - After a cookie expires it will automatically remove itself from the client's computer.
- When a web site writes a cookie to a client's computer, only that web site can later read the cookie's value.
 - It is the browser's responsibility to keep track of which web site created the cookie, and only allow cookies to be read by the proper web sites.
- It is important to note that all modern browsers provide users with an option not to accept cookies on their computers; however they do accept cookies by default.

Using Cookies

- If you wanted to store **multiple items of information** you could use several different cookies.
- However it more sensible to create a logical structure for your cookies using **keys**.
 - Each cookie can optionally have a set of keys.
 - Each key can be used to save an item of information.
 - For example, if you wanted to use cookies to save information about visitors it would be sensible to create a single cookie named `UserInfo` and a set of keys for holding the user's name, the last time he accessed your web pages, etc.
- Most browsers support cookies of up to 4096 bytes.
 - Because of this small limit, cookies are best used to store small amounts of data.
 - Browsers also impose limitations on how many cookies your site can store on the user's computer. Most browsers allow only 20 cookies per site; if you try to store more, the oldest cookies are discarded.
 - Some browsers also put an absolute limit, usually 300, on the number of cookies they will accept from all sites combined.

Using Cookies

Reading Cookies

- Cookies are stored and read using **HTTP headers**
 - When a client requests a web page from a server, it not only sends the URL of the web page requested but also some additional information.
 - This information consists of useful facts about the client e.g. what browser they are using, what operating system, and the cookies present on the client. This information is called the HTTP headers.
- Cookies can be read using the **Request object**:
`Request.Cookies["cookieName"]["keyname"];`
- Each cookie can have **zero to many keys**.
 - If a cookie or its keys do not exist on a client's computer (i.e. if they have never been written, or the client has turned off cookies in his browser), then the Request object will return an empty string.

Using Cookies

Writing Cookies

- Cookies can be written to the client's computer through the HTTP headers using the **Response object**:
`Response.Cookies["cookieName"]["keyName"] = data;`
- Cookies can contain any single data type
 - e.g. strings, numbers, dates, currencies, and Boolean values, but cannot store arrays or objects.
- If we wanted to create a cookie called `UserInfo` on the client's computer to keep track of a visitor's name and the last time they visited the web site then we would use the following ASP code:

```
Response.Cookies["UserInfo"]["Name"] = "Dave";  
Response.Cookies["UserInfo"]["LastVisit"] =  
    DateTime.Now.ToString();
```

Using Cookies

Writing Cookies

- When a cookie **expires** it deletes itself from the client's computer.
- When creating cookies, you can specify when the cookie should expire using the **Expires property**.
 - If you do not specify this value, then the cookie is set to expire when the user closes the browser.
 - If you want to maintain state for the duration of the user's visit to your site then the default value is ideal.
 - However, cookies are often used to persist state over days, weeks or years.
 - To accommodate for this lengthy a period the Expires property must be set. For example, the following would set the cookie to expire in one week's time:

```
Response.Cookies["UserInformation"].Expires =  
    DateTime.Now.AddDays(7);
```

Using Session State

- **Session State** enables you to preserve the values of variables across page requests for a particular user.
 - Session State is typically used for storing shopping carts, user preferences, or user security information, such as usernames and passwords.
- Items that you add to Session State are **stored in the Web Server's memory**.
 - You can add almost any type of object to Session State, including strings, integers, ArrayLists, DataSets, and DataTables.
 - The only requirement is that the object be serializable.
- Each variable stored in the Session object is referred to as a **session variable** and can be created as follows:

```
Session[sessionVariableName] = value;
```

- For example:

```
Session["WelcomeMessage"] = "Hello world";
```


Using Session State

- To read the value of your Session variable in another ASP page on your site you do the following:
`SomeVariable = Session[sessionVariableName];`
- Because each user's Session is created and stored in the Web server's memory, it is important to have this memory **freed up** when the visitor leaves your site.
- Due to the nature of the client server model you cannot determine when a visitor leaves your site.
 - However you can keep track of the last time a specific user accessed your site.
 - If a specific user has not accessed your site for a particular amount of time the Session is freed from the Web server's memory.
 - This amount of time can be set as follows (e.g. 5 minutes):
`Session.Timeout = 5;`

Using Session State

- By default the session timeout is set to 20 minutes.
- If you want to abandon the users Session explicitly (e.g. by providing a LogOut button) then you use the following:

```
Session.Abandon;
```

- You should be careful not to put too many variables into the Session object.
- All visitors to your site will create a set of Session variables each occupying their own area of Web server's memory.
 - If your site is very busy, with a lot of concurrent users, this will result in large amounts of Web server memory being used and this will severely affect the performance of your site.
- As Session Variables use a cookie to store the session ID (and hence identify the owner of the session), cookies must be turned on for Session Variables to work.

Using Application State

- **Application State** variables enables you to preserve the values of variables across page requests for the entire web site.
- Items that you add to Application State are stored in the Web Server's memory.
- Each variable stored in the Application object is referred to as a application variable and can be created by:

```
Application[applicationVariableName] = value;
```

- For example:

```
Application["WelcomeMessage"]= "Hello world";
```

- To read the value of your Application variable in another ASP.NET page on you site you do the following:

```
SomeVariable = Application[applicationVariableName];
```

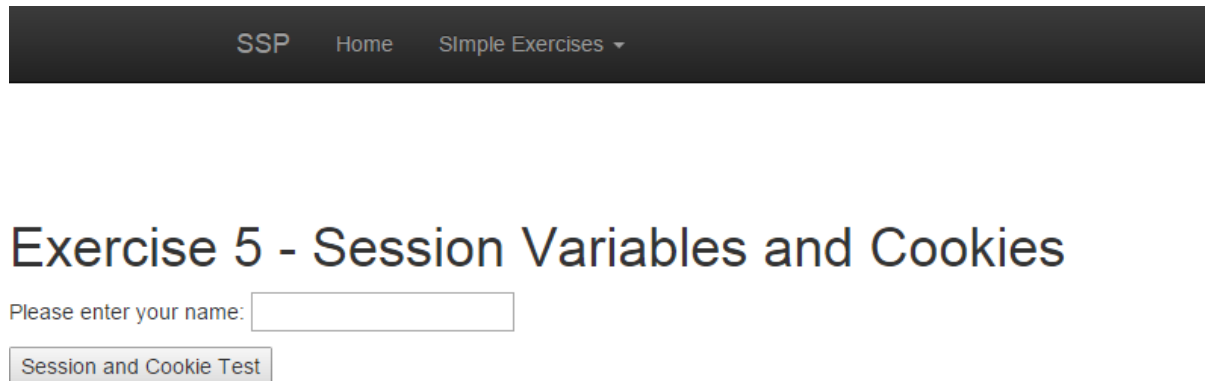
Using Application State

- Because **only one instance of the Application object exists for the entire web site**, it should only be used to store information that is specific to the entire web site.
- Because **only one application object is shared among all users of your web site** you must take care when updating application variables to ensure two users don't try to access the same variable at the same time.
 - You do this by locking the application object as follows:

```
Application.Lock;  
Application["WelcomeMessage"]= "Hello world";  
Application.Unlock;
```

Ex 5 – Cookies and Session Variables

- Add a new form to your existing ASP.NET web application.
 - Name the new Web Form **exercise5.aspx**.
- Add another new form to your existing ASP.NET web application.
 - Name the new Web Form **sessiontest.aspx**.
- On the web form exercise5.aspx add Label boxes, Text boxes and Buttons as shown below:



SSP Home Simple Exercises ▾

Exercise 5 - Session Variables and Cookies

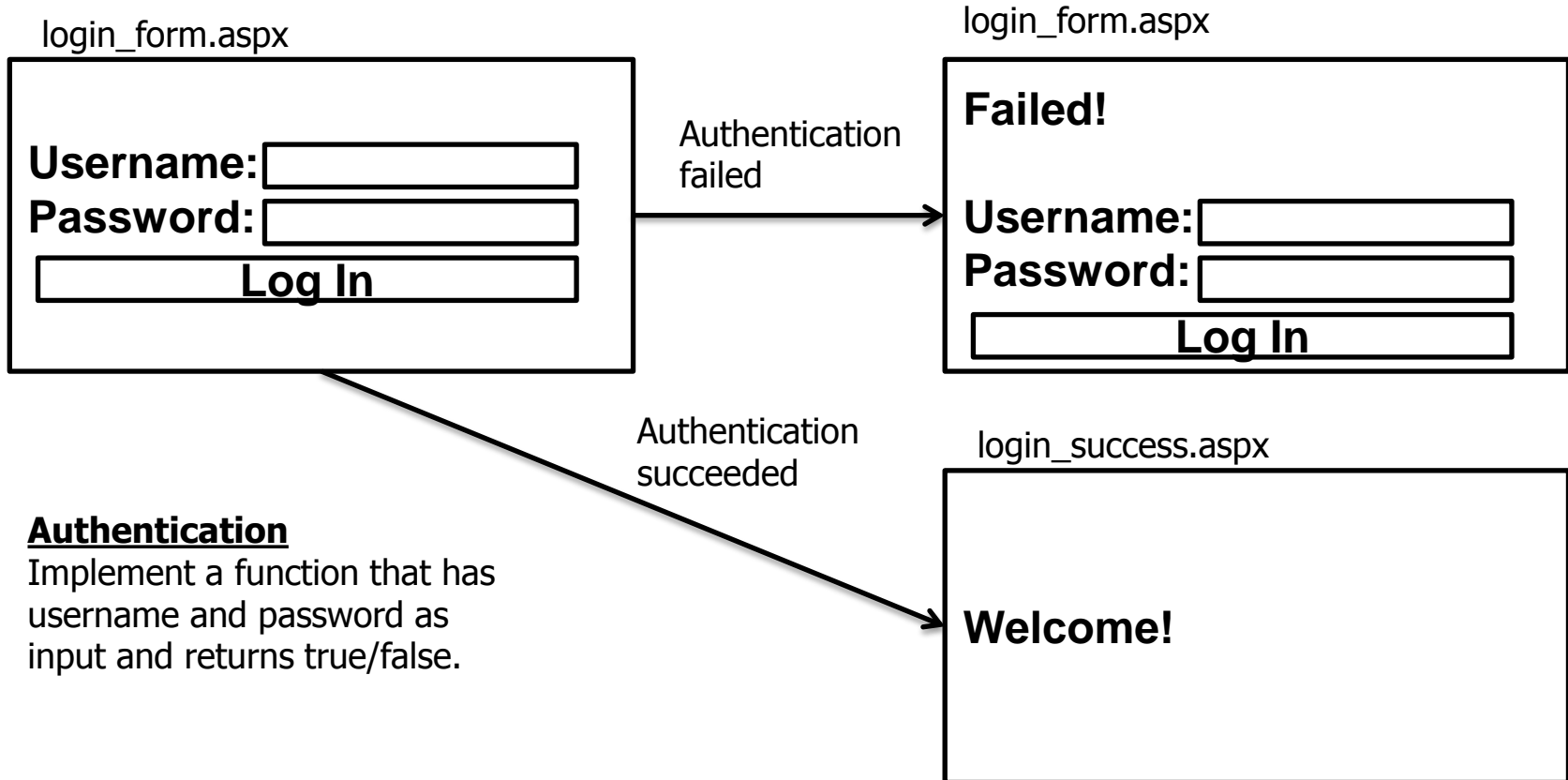
Please enter your name:

Ex 5 – Cookies and Session Variables

- Add code to the **command button** which reads the information from the textbox and passes it to a new page via a **session variable** and via a **cookie**.
 - Hint: to force the web site to go to another page rather than post back use `Response.Redirect("sessiontest.aspx")`.
- Another **cookie** should be added to pass the current date and time to the new page.
- Code should be added to set the **session timeout** and change the **cookie expiration date**.
- The form `sessiontest.aspx` should display the information passed to it as shown:



Ex 5a – Complete login



Authentication

Implement a function that has username and password as input and returns true/false.

Part 2

Add the following completely independent feature: Have a set of random numbers/letters by the login form and a textbox for the user to type them. If incorrect, go back to the login form WITHOUT having called the authentication function.