COMP 3111: Software Engineering

Tutorial and Lab Notes
State Management in ASP.NET

Most web applications will require variables that need to be maintained across a series of requests or shared between multiple clients of an application. These variables are referred to as *state*. Managing state in web applications is challenging because the HTTP protocol is inherently stateless (i.e., it does not provide a way for the web server to identify that a series of requests comes from the same client). This makes it difficult for the web server to maintain and associate state with an individual client. In addition to individual client state, many applications need to store and retrieve application-level state information that is global to all clients of the web application.

ASP.NET provides several options for storing state depending on, among other considerations, where you want to store the data, how much data you want to store, how long you want it to be retained and from where you want the data to be accessible. In most cases, the state is stored in a dictionary collection as *name-value pairs* (referred to as *key-value pairs*) where each item in the collection is indexed with a unique string name (the *key*), which has an associated value (the *value*).

Cookies

Cookies are small files created on the client's computer (or in the web browser's memory if they are temporary) and used to store simple string information. While cookies can be retained between visits, they are available only on the computer on which they were originally stored. Since the storing of cookies can be disabled on a browser, using them may cause problems for web applications that require them.

The System.Net namespace needs to be imported to use cookies. Cookies are set using the `Response` object and retrieved using the `Request` object. To set a cookie, simply create a new `System.Net.HttpCookie` object. You can then fill it with string information using the key-value pair pattern and attach it to the current web response. A cookie persists until the user closes the browser and is sent with every request. To create a longer-lived cookie, an expiration date can be set for the cookie using the cookie object's `Expires` method.

An example use of cookies is to store the fact that a user has been verified to access certain web pages on a web site (i.e., has successfully logged in).

View State

View state is used for storing data within a web page that is accessible only to the page and is stored only as long as the page is active. For example, the values of the web controls in a web form and their corresponding ID's that would otherwise be lost due to a postback because they do not post with the form are stored in view state. ASP.NET stores view state information in a hidden form field, called `_ViewState` as a collection of key-value pairs.

To add your own data, such as simple data types and custom objects[1], to view state you can use the built-in page property called `ViewState`. For example, the statement

```
ViewState.Add("Name", "Fred")
```

places the value "Fred" into view state and gives it the unique, descriptive key "Name". If there is currently no item with the key "Name" in view state, then a new item is added; otherwise, "Fred" replaces the existing value of "Name".

To retrieve a value from a page's `_ViewState` collection, you use the descriptive key (i.e., the name) that you assigned it. It is also necessary to cast the retrieved value to the appropriate data type since view state stores all items as standard objects, which allows it to handle many different data types. For example, to retrieve the value of "Name" from the `_ViewState` collection and assign it to the Text property of a Label control whose ID is lblName, you would use the following statement

```
if ViewState["Name"] != null
{
   lblName.Text = ViewState["Name"].ToString();
}
```

[1] To be able to store objects in View State, the objects must be serializable. To make an object serializable, the `<Serializable>` statement must be placed before its class declaration.

Note that if you attempt to look up a key that is not present in the `_ViewState` collection, you will get a NullReferenceException. To guard against this, you should always check for a null value as in the above example before attempting to retrieve and cast view state data.

An example use of view state is to store the values of text boxes on a web form so that if the user reloads the page, any values already entered are not lost.

Session State

Session sate is used for storing data that you want to keep private for a user of your web application (i.e., not share with other users), but still be able to access from different web pages. Items in session state last until the user's browser session ends, usually due to a timeout. ASP.NET session state consists of key-value pairs, which are stored as a collection associated with the `HttpSessionState` class. This collection of key-value pairs can be manipulated using the following methods of the `Session` object[2].

`Abandon`	cancel the current session
`Add`	add a variable to the session state
`Clear/RemoveAll`	remove all variables from the session state
`Remove`	remove a variable from the session state
`RemoveAt`	remove a specific variable from the session state based on its index
`ToString`	return a string that represents a value in the session state. (Used when a string value is required rather than an object reference.)

The Session object also provides the following properties.

`Count`	the number of variables stored in the session state
`IsCookielessSession`	Indicates whether this session is tracked with a cookie or a modified URL
`IsNewSession`	Indicates whether this session was just created from the current request
`Keys`	a collection of all variable names used to access session state values
`SessionID`	a string containing the unique session ID of the current session
`Timeout`	an integer value representing the current session timeout setting

Each user of a web application has his own `Session` object instance, identified by a session ID, which is a unique identifier generated by the server for each instance of a user session.

Session state must be enabled before it can be used; however, it is enabled by default. The `enableSessionState` attribute of the `@ Page` directive can be used to control whether or not session state is enabled for a web page. Moreover, in Visual Studio you can also enable (or delay enabling) session state by setting the enableSessionState property of the Document object in the Properties window.

An example use of session state is to store a user's shopping cart in memory as he browses from page to page in an E-commerce web site.

Application State

Application state is used for storing data that you want to share among multiple users of your web application. Items in application state last until the application or server is restarted or until the application domain refreshes itself. ASP.NET application state consists of key-value pairs, which are stored as a collection associated with the `HttpApplicationState` class. This collection of key-value pairs can be manipulated using the following methods of the `Application` object[3].

`Add`	add a variable to the application state
`Clear/RemoveAll`	delete all variables in the application state
`Get`	return a value from the application state by a variable name or by an index
`GetKey`	return the name of a variable based on a supplied index
`Lock`	lock the application state (i.e., Application object) for update
`Remove`	remove a variable from the application state

[2] A `Session` object is an instance of the `HttpSessionState` class and is exposed as the Session property of the `Page` class.

[3] An `Application` object is an instance of the `HttpApplicationState` class and is exposed as the Application property of the `Page` class. Since every ASP.NET page inherits from the `Page` class, an `Application` object can be accessed as if it were an inherent property of a page.

`RemoveAt`	remove a specific variable from the application state based on its index
`Set`	update a value in the application state by a variable name or by an index
`ToString`	return a string that represents a value in the application state. (Used when a string value is required rather than an object reference.)
`Unlock`	unlock the application state (i.e., Application object)

The Application object also provides the following properties:

`AllKeys`	return a collection of all of the variable names by which application state values can be accessed
`Count`	return the number of variables stored in the application state

Since application state (i.e., the application variables) is shared among multiple users of a web application, you need to ensure that no two users can update the application state information simultaneously. The `Application` object provides the `Lock` and `Unlock` methods for this purpose. Since no other users can access the `Application` object while it is locked, the time that you lock the `Application` object in your web application should be as short as possible.

The following are some points to note regarding application state.

- Any add/update/delete of application variables will be available to all browser windows of the same web application.
- Application state lasts only as long as the web application is running. Stopping the web application or the web server results in the destruction of the application state.

An example use of application state is to keep track of the number of online users in a web site.

Creating, Viewing and Updating Session and Application Variables

We will demonstrate how to create session and application variables and the difference in their use by creating a web form through which you can create, view and update these variables.

1. Start Visual Studio and create a new web application called “StateManagement”.
2. Add a new web form named `StateManagementExample.aspx`.
3. Add the controls shown in Figure 1 and set their properties as shown in Table 1.[4]

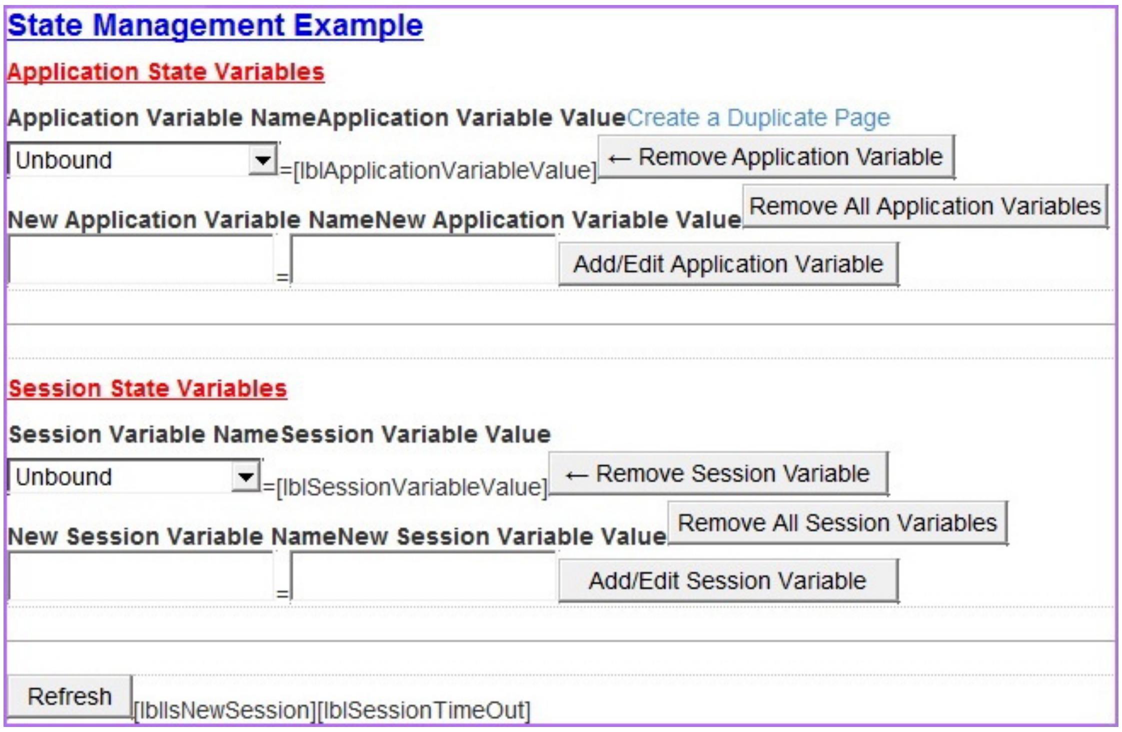

State Management Example
Application State Variables
Application Variable Name
Application Variable Value
Create a Duplicate Page
Unbound
=[lblApplicationVariableValue]
← Remove Application Variable
New Application Variable Name
New Application Variable Value
Remove All Application Variables
=
Add/Edit Application Variable
Session State Variables
Session Variable Name
Session Variable Value
Unbound
=[lblSessionVariableValue]
← Remove Session Variable
New Session Variable Name
New Session Variable Value
Remove All Session Variables
=
Add/Edit Session Variable
Refresh
[lblIsNewSession][lblSessionTimeOut]

Figure 1: `StateManagementExample.aspx` web form controls.

[4] To save time, you can download the `StateManagement.zip` web application from the Lab and Tutorial Schedule module of the course web page. You should study and understand the code in the `StateManagementExample.aspx.cs` file *before continuing with testing as described on page 9.*

Table 1: `StateManagementExample.aspx` web form controls and their properties.

Line	Control	Property	Value
1	Label1	Text	Application Variable Name
	Label2	Text	Application Variable Value
	Hyperlink1	Text	Create a Duplicate Page
		NavigateUrl	*Click the ellipse (…) button and navigate to the file StateManagementExample.aspx*
2	DropDownList1	ID	ddlApplicationVariableName
		AutoPostBack	True
	Label3	Text	=
	Label4	ID	lblApplicationVariableValue
		Text	(blank)
	Button1	ID	btnRemoveApplicationVariable
		Text	← Remove Application Variable
3	Label5	Text	New Application Variable Name
	Label6	Text	New Application Variable Value
	Button2	ID	btnRemoveAllApplicationVariables
		Text	Remove All Application Variables
4	TextBox1	ID	txtApplicationVariableName
	Label7	Text	=
	TextBox2	ID	txtApplicationVariableValue
	Button3	ID	btnAddEditApplicationVariable
		Text	Add/Edit Application Variable
5	Label8	Text	Session Variable Name
	Label9	Text	Session Variable Value
6	DropDownList2	ID	ddlSessionVariableName
		AutoPostBack	True
	Label10	Text	=
	Label11	ID	lblSessionVariableValue
		Text	(blank)
	Button4	ID	btnRemoveSessionVariable
		Text	← Remove Session Variable
7	Label12	Text	New Session Variable Name
	Label13	Text	New Session Variable Value
	Button5	ID	btnRemoveAllSessionVariables
		Text	Remove All Session Variables
8	TextBox3	ID	txtSessionVariableName
	Label14	Text	=
	TextBox4	ID	txtSessionVariableValue
	Button6	ID	btnAddEditSessionVariable
		Text	Add/Edit Session Variable
9	Button7	ID	btnRefresh
		Text	Refresh
	Label15	ID	lblIsNewSession
		Text	(blank)
	Label16	ID	lblSessionTimeOut
		Text	(blank)

The next three steps will define the event handlers for the buttons in the Application State Variables section of the web form.

4. Double-click the "← Remove Application Variable" button and add the following code[5] into the "btnRemoveApplicationVariable_Click" event handler as shown in Figure 2. This code removes the application variable currently selected in the dropdown list[6].

```
Application.Remove(ddlApplicationVariableName.SelectedItem.ToString());
ShowApplicationVariables();
```

5. Double click the "Remove All Application Variables" button and add the following code into the "btnRemoveAllApplicationVariables_Click" event handler as shown in Figure 2. This code deletes all application variables.

```
Application.RemoveAll();
ShowApplicationVariables();
```

6. Double click the "Add/Edit Application Variable" button and add the following code into the "btnAddEditApplicationVariable_Click" event handler as shown in Figure 2. This code either adds a new application variable and its value or it changes the value of an existing application variable.

```
if (txtApplicationVariableName.Text != "")
{
   Application.Lock();
   Application[txtApplicationVariableName.Text] =
      txtApplicationVariableValue.Text;
   Application.UnLock();
   ShowApplicationVariables();
}
```

We will now insert code to display the value of the application variable selected in the dropdown list in the Application State Variables section of the web form.

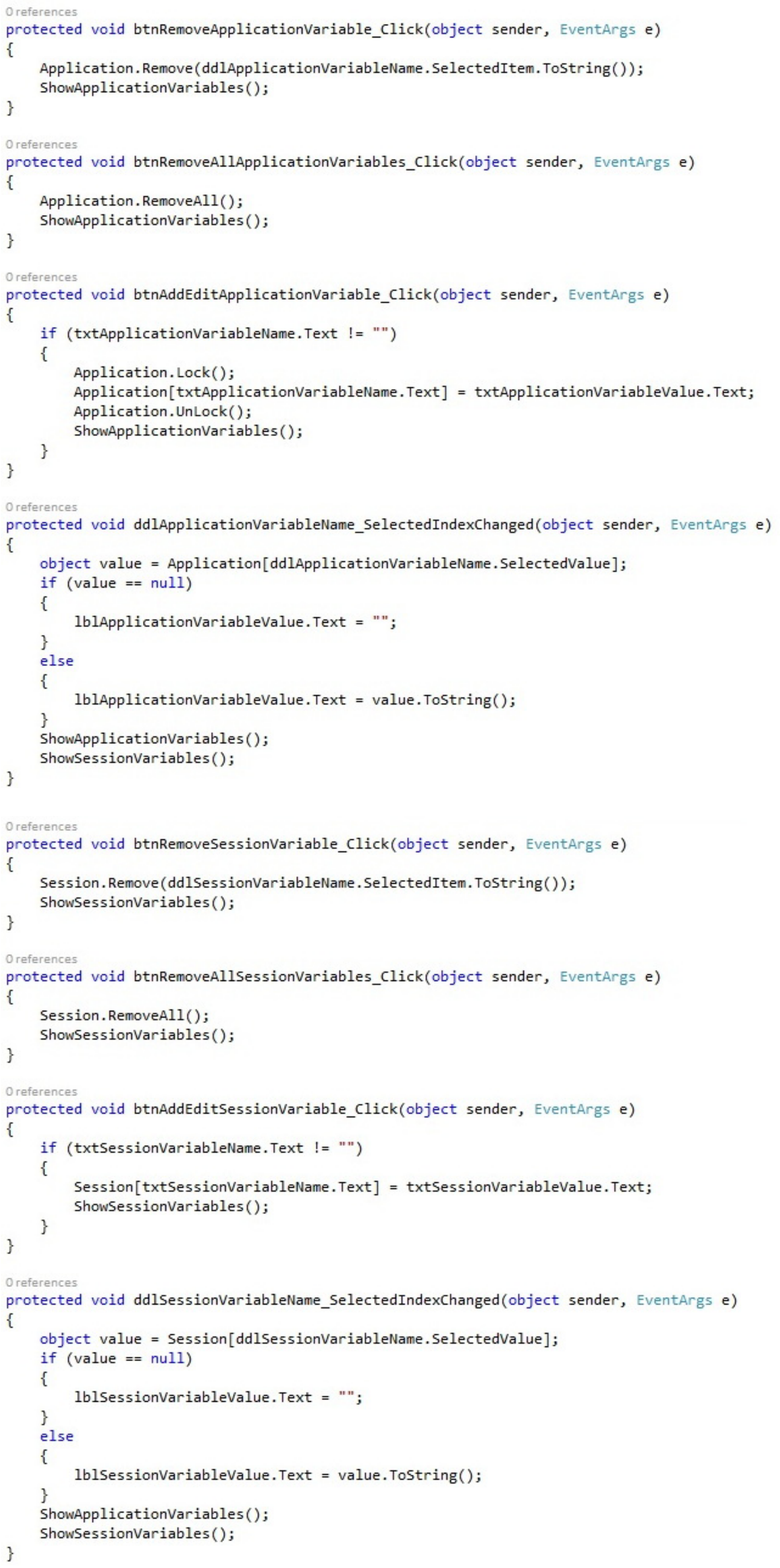
7. In the `Properties` window for the "ddlApplicationVariableName" DropDownList control[7] select the Events tab, double-click the SelectedIndexChanged event and add the following code into the "ddlApplicationVariableName_SelectedIndexChanged" even handler as shown in Figure 2. This code displays the value (as a string) of the application variable selected from the dropdown list.

```
object value = Application[ddlApplicationVariableName.SelectedValue];
if (value == null)
{
   lblApplicationVariableValue.Text = "";
}
else
{
   lblApplicationVariableValue.Text = value.ToString();
}
ShowApplicationVariables();
ShowSessionVariables();
```

8. Create the event handlers for the buttons "btnRemoveSessionVariable", "btnRemoveAllSessionVariables" and "btnAddEditSessionVariable" and the dropdown list "ddlSessionVariableName" in a way similar to the preceding three buttons and dropdown list. The code is similar to that for Application state except that you need to invoke the state management methods of the `Session` object (i.e., substitute "Session" for "Application" wherever it appears in the code). Furthermore, you need to remove the lines "Application.Lock()" and "Application.UnLock()" from the code for the "btnAddEditSessionVariable_Click" event handler since there is no need to lock the `Session` object as it cannot be shared among different browser sessions. The required code for the event handlers is shown in Figure 2.
9. Double-click the "Refresh" button and add the following code into the "btnRefresh_Click" event handler. This code will force a postback to refresh the page.

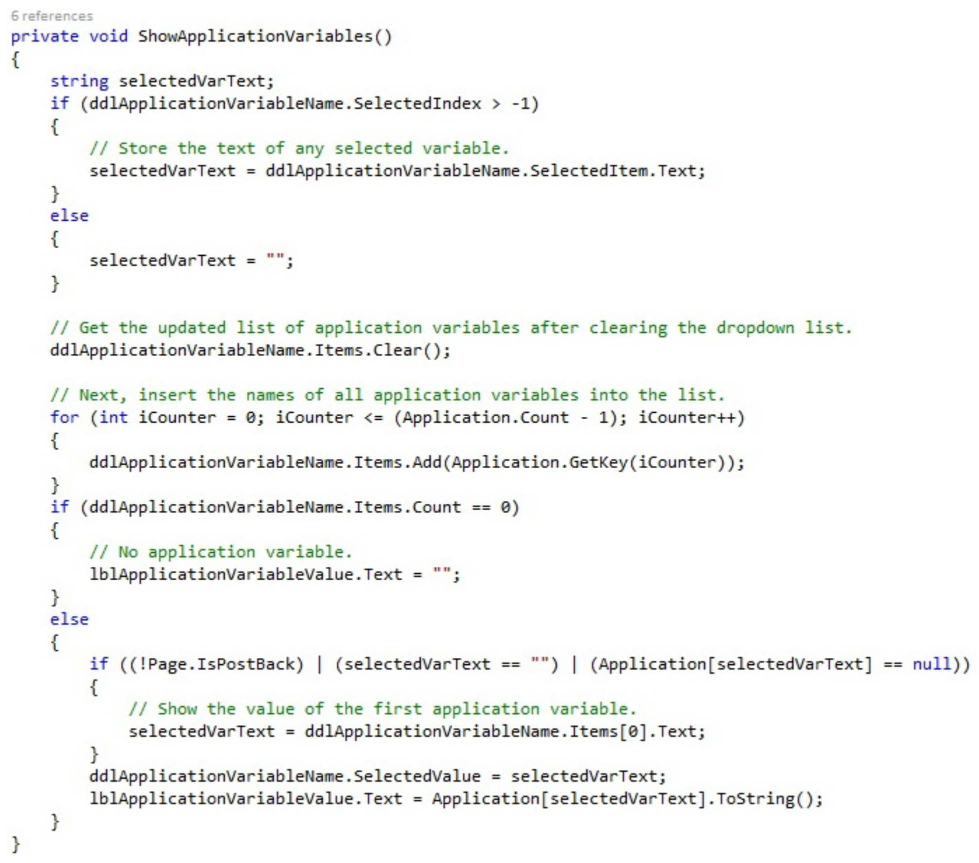
```
Server.Transfer("StateManagementExample.aspx");
```

[5] You can download the file "StateManagement-Download.zip" containing all the code for this lab from the Lab and Tutorial Schedule module of the course web page.
[6] Note that the procedure ShowApplicationVariables() will be marked as being undeclared. It will be declared in step 0.
[7] This is the ID of the DropDownList control in the application state section of the web form.

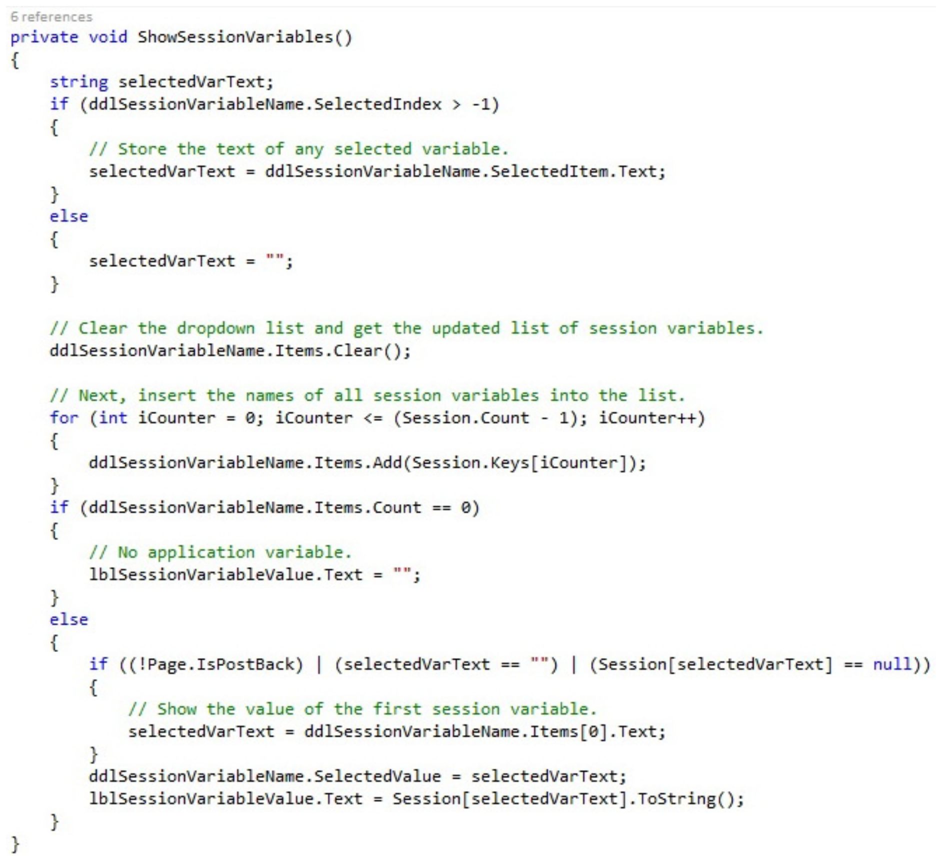


```
0 references
protected void btnRemoveApplicationVariable_Click(object sender, EventArgs e)
{
    Application.Remove(ddlApplicationVariableName.SelectedItem.ToString());
    ShowApplicationVariables();
}

0 references
protected void btnRemoveAllApplicationVariables_Click(object sender, EventArgs e)
{
    Application.RemoveAll();
    ShowApplicationVariables();
}

0 references
protected void btnAddEditApplicationVariable_Click(object sender, EventArgs e)
{
    if (txtApplicationVariableName.Text != "")
    {
        Application.Lock();
        Application[txtApplicationVariableName.Text] = txtApplicationVariableValue.Text;
        Application.UnLock();
        ShowApplicationVariables();
    }
}

0 references
protected void ddlApplicationVariableName_SelectedIndexChanged(object sender, EventArgs e)
{
    object value = Application[ddlApplicationVariableName.SelectedValue];
    if (value == null)
    {
        lblApplicationVariableValue.Text = "";
    }
    else
    {
        lblApplicationVariableValue.Text = value.ToString();
    }
    ShowApplicationVariables();
    ShowSessionVariables();
}


0 references
protected void btnRemoveSessionVariable_Click(object sender, EventArgs e)
{
    Session.Remove(ddlSessionVariableName.SelectedItem.ToString());
    ShowSessionVariables();
}

0 references
protected void btnRemoveAllSessionVariables_Click(object sender, EventArgs e)
{
    Session.RemoveAll();
    ShowSessionVariables();
}

0 references
protected void btnAddEditSessionVariable_Click(object sender, EventArgs e)
{
    if (txtSessionVariableName.Text != "")
    {
        Session[txtSessionVariableName.Text] = txtSessionVariableValue.Text;
        ShowSessionVariables();
    }
}

0 references
protected void ddlSessionVariableName_SelectedIndexChanged(object sender, EventArgs e)
{
    object value = Session[ddlSessionVariableName.SelectedValue];
    if (value == null)
    {
        lblSessionVariableValue.Text = "";
    }
    else
    {
        lblSessionVariableValue.Text = value.ToString();
    }
    ShowApplicationVariables();
    ShowSessionVariables();
}
```

Figure 2: Code for application and session variable event handlers.



```
6 references
private void ShowApplicationVariables()
{
    string selectedVarText;
    if (ddlApplicationVariableName.SelectedIndex > -1)
    {
        // Store the text of any selected variable.
        selectedVarText = ddlApplicationVariableName.SelectedItem.Text;
    }
    else
    {
        selectedVarText = "";
    }

    // Get the updated list of application variables after clearing the dropdown list.
    ddlApplicationVariableName.Items.Clear();

    // Next, insert the names of all application variables into the list.
    for (int iCounter = 0; iCounter <= (Application.Count - 1); iCounter++)
    {
        ddlApplicationVariableName.Items.Add(Application.GetKey(iCounter));
    }
    if (ddlApplicationVariableName.Items.Count == 0)
    {
        // No application variable.
        lblApplicationVariableValue.Text = "";
    }
    else
    {
        if ((!Page.IsPostBack) | (selectedVarText == "") | (Application[selectedVarText] == null))
        {
            // Show the value of the first application variable.
            selectedVarText = ddlApplicationVariableName.Items[0].Text;
        }
        ddlApplicationVariableName.SelectedValue = selectedVarText;
        lblApplicationVariableValue.Text = Application[selectedVarText].ToString();
    }
}
```



```
6 references
private void ShowSessionVariables()
{
    string selectedVarText;
    if (ddlSessionVariableName.SelectedIndex > -1)
    {
        // Store the text of any selected variable.
        selectedVarText = ddlSessionVariableName.SelectedItem.Text;
    }
    else
    {
        selectedVarText = "";
    }

    // Clear the dropdown list and get the updated list of session variables.
    ddlSessionVariableName.Items.Clear();

    // Next, insert the names of all session variables into the list.
    for (int iCounter = 0; iCounter <= (Session.Count - 1); iCounter++)
    {
        ddlSessionVariableName.Items.Add(Session.Keys[iCounter]);
    }
    if (ddlSessionVariableName.Items.Count == 0)
    {
        // No application variable.
        lblSessionVariableValue.Text = "";
    }
    else
    {
        if ((!Page.IsPostBack) | (selectedVarText == "") | (Session[selectedVarText] == null))
        {
            // Show the value of the first session variable.
            selectedVarText = ddlSessionVariableName.Items[0].Text;
        }
        ddlSessionVariableName.SelectedValue = selectedVarText;
        lblSessionVariableValue.Text = Session[selectedVarText].ToString();
    }
}
```

Figure 3: Code to update and display application and session variables.

10. Add the following two new private methods into the code-behind file as shown in Figure 3. These two methods update the list of application and session variables and their displayed values, respectively[8].

```
private void ShowApplicationVariables()
{
   string selectedVarText;
   if (ddlApplicationVariableName.SelectedIndex > -1)
   {
      // Store the text of any selected variable.
      selectedVarText = ddlApplicationVariableName.SelectedItem.Text;
   }
   else
   {
      selectedVarText = "";
   }
   // Get the updated list of application variables after clearing the
      dropdown list.
   ddlApplicationVariableName.Items.Clear();

   // Next, insert the names of all application variables into the list.
   for (int iCounter = 0; iCounter <= (Application.Count - 1); iCounter++)
   {
      ddlApplicationVariableName.Items.Add(Application.GetKey(iCounter));
   }
   if (ddlApplicationVariableName.Items.Count == 0)
   {
      // No application variable.
      lblApplicationVariableValue.Text = "";
   }
   else
   {
      if ((!Page.IsPostBack) | (selectedVarText == "") |
         (Application[selectedVarText] == null))
      {
         // Show the value of the first application variable.
         selectedVarText = ddlApplicationVariableName.Items[0].Text;
      }
      ddlApplicationVariableName.SelectedValue = selectedVarText;
      lblApplicationVariableValue.Text =
         Application[selectedVarText].ToString();
   }
}
```

The above code refreshes the list of application variables because they may be added/deleted/updated by other browsers that access the web page. Furthermore, since other browsers may remove an application variable, a check for its existence is required before displaying its value.

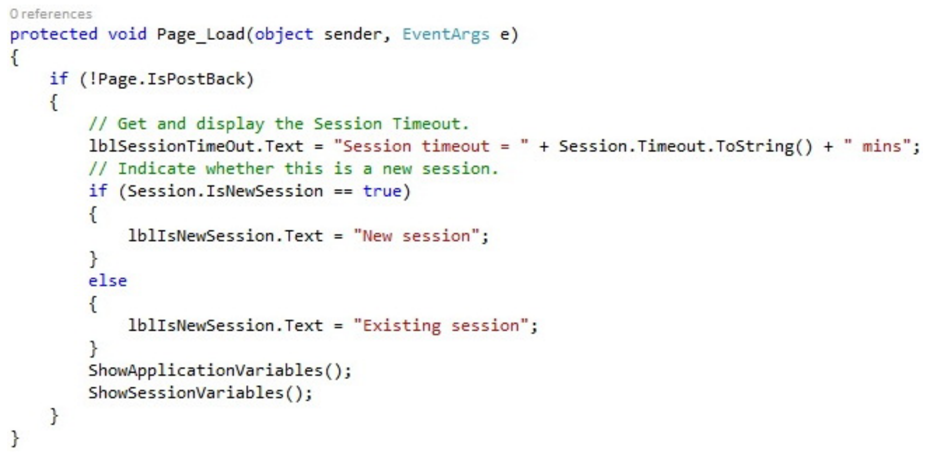
```
private void ShowSessionVariables()
{
   string selectedVarText;
   if (ddlSessionVariableName.SelectedIndex > -1)
   {
      // Store the text of any selected variable.
      selectedVarText = ddlSessionVariableName.SelectedItem.Text;
   }
   else
   {
      selectedVarText = "";
   }
   // Clear the dropdown list and get the updated list of session
      variables.
```

[8] The code to display session variables and their values is similar to that for application variables except for the blue italic line. It is possible to merge the code into one procedure using an ASP.NET User Control.

```
   ddlSessionVariableName.Items.Clear();
   // Next, insert the names of all session variables into the list.
   for (int iCounter = 0; iCounter <= (Session.Count - 1); iCounter++)
   {
      ddlSessionVariableName.Items.Add(Session.Keys[iCounter]);
   }
   if (ddlSessionVariableName.Items.Count == 0)
   {
      // No application variable.
      lblSessionVariableValue.Text = "";
   }
   else
   {
      if ((!Page.IsPostBack) | (selectedVarText == "") |
         (Session[selectedVarText] == null))
      {
         // Show the value of the first session variable.
         selectedVarText = ddlSessionVariableName.Items[0].Text;
      }
      ddlSessionVariableName.SelectedValue = selectedVarText;
      lblSessionVariableValue.Text = Session[selectedVarText].ToString();
   }
}
```

11. In the Page_Load event handler, insert the following code, as shown in Figure 4, which retrieves and displays the Timeout value of the Session object, indicates whether this is a new session and updates the dropdown lists of the application and session variables.

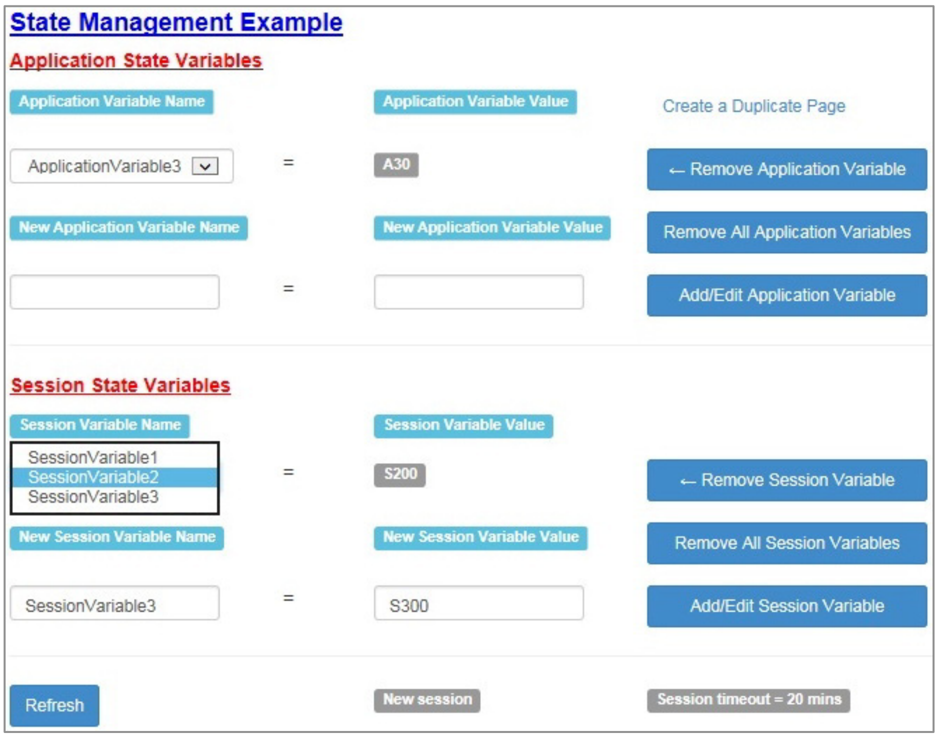
```
if (!Page.IsPostBack)
{
   // Get and display the Session Timeout.
   lblSessionTimeOut.Text = "Session timeout = " +
      Session.Timeout.ToString() + " mins";
   // Indicate whether this is a new session.
   if (Session.IsNewSession == true)
   {
      lblIsNewSession.Text = "New Session";
   }
   else
   {
      lblIsNewSession.Text = "Existing session";
   }
   ShowApplicationVariables();
   ShowSessionVariables();
}
```



```
0 references
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        // Get and display the Session Timeout.
        lblSessionTimeOut.Text = "Session timeout = " + Session.Timeout.ToString() + " mins";
        // Indicate whether this is a new session.
        if (Session.IsNewSession == true)
        {
            lblIsNewSession.Text = "New session";
        }
        else
        {
            lblIsNewSession.Text = "Existing session";
        }
        ShowApplicationVariables();
        ShowSessionVariables();
    }
}
```

Figure 4: Code to display session information and update application and session variables.

Save your work, set the “StateManagementExample .aspx” web form as the start page and view the web form in a browser. Add, update, delete and retrieve application and session variables to test the page as shown in Figure 5.


State Management Example
Application State Variables
Application Variable Name
Application Variable Value
Create a Duplicate Page
← Remove Application Variable
New Application Variable Name
New Application Variable Value
Remove All Application Variables
Add/Edit Application Variable
Session State Variables
Session Variable Name
Session Variable Value
← Remove Session Variable
New Session Variable Name
New Session Variable Value
Remove All Session Variables
Add/Edit Session Variable
Refresh

ApplicationVariable3 = A30

SessionVariable1
SessionVariable2 = S200
SessionVariable3

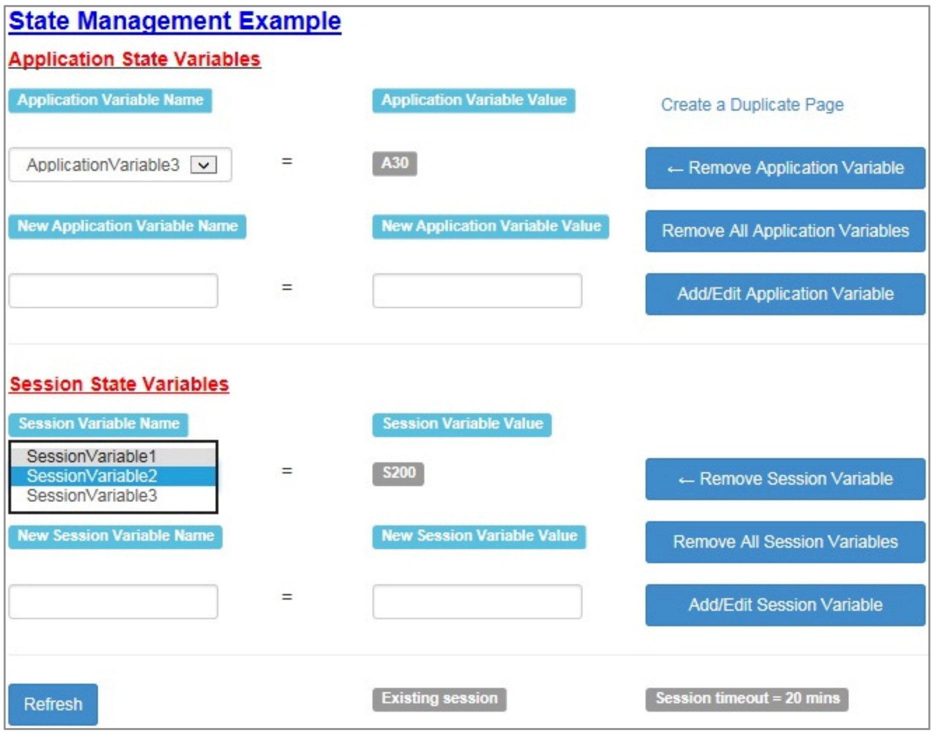
SessionVariable3 = S300

New session

Session timeout = 20 mins

Figure 5: First browser window for first session.

Right-click the hyperlink “Create a Duplicate Page” and select “Open in New Window” to open another browser window for the web form. You will see the application variables, which are shared by all browsers of the web form. The same session variables also can be found in the new browser window as shown in Figure 6 because both browser windows are in the same session as indicated by the message “Existing session” at the bottom of the page. Add, update and delete application and session variables in one browser window and you will see the change in the other browser window[9].


State Management Example
Application State Variables
Application Variable Name
Application Variable Value
Create a Duplicate Page
← Remove Application Variable
New Application Variable Name
New Application Variable Value
Remove All Application Variables
Add/Edit Application Variable
Session State Variables
Session Variable Name
Session Variable Value
← Remove Session Variable
New Session Variable Name
New Session Variable Value
Remove All Session Variables
Add/Edit Session Variable
Refresh

ApplicationVariable3 = A30

SessionVariable1
SessionVariable2 = S200
SessionVariable3

Existing session

Session timeout = 20 mins

Figure 6: Second browser window for first session.

[9] You may need to “refresh” the display of the application and session variables to see any changes by clicking the refresh button.

Open a new session browser window by selecting "File→New Session" from the Internet Explorer menu. Enter the URL of the web form to open it. You will not find any session variables in the new browser window as shown in Figure 7 because the new browser is inside another session as indicated by the message "New session" at the bottom of the page.

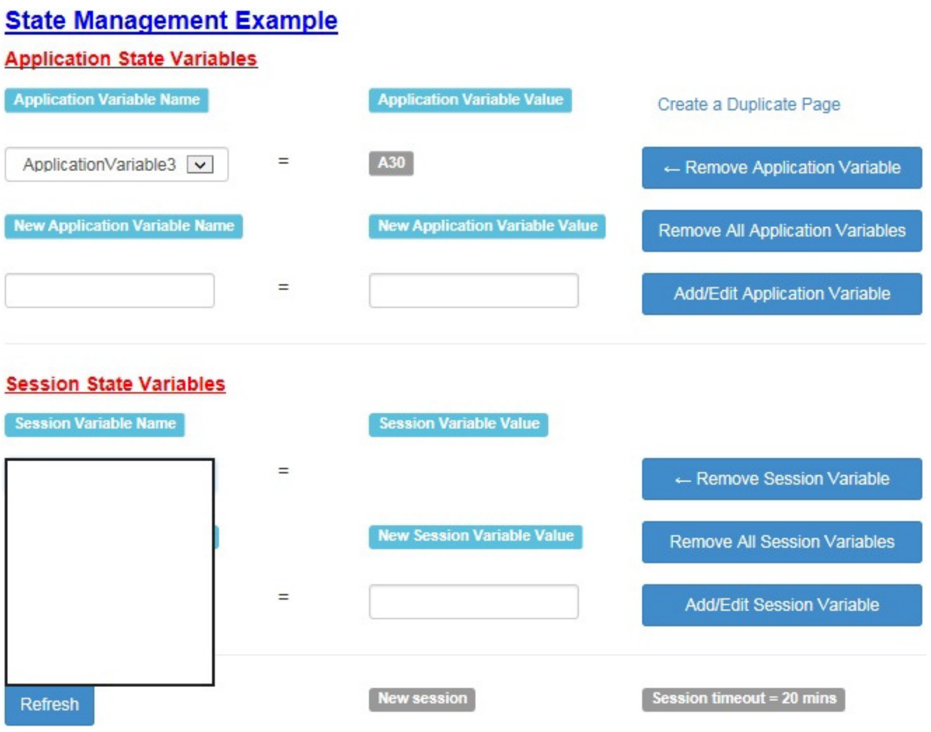

State Management Example
Application State Variables
Application Variable Name
Application Variable Value
Create a Duplicate Page
ApplicationVariable3
=
A30
← Remove Application Variable
New Application Variable Name
New Application Variable Value
Remove All Application Variables
=
Add/Edit Application Variable
Session State Variables
Session Variable Name
Session Variable Value
=
← Remove Session Variable
New Session Variable Value
Remove All Session Variables
=
Add/Edit Session Variable
Refresh
New session
Session timeout = 20 mins

Figure 7: New browser window for second session.