Chapter 11

User Filter State

When users visit a search page (or an “add, edit and delete” page with a set of search filters above the grid), each user will enter search criteria, drill down on a search result and then return to the search page. When the user returns to search again, there is a strong expectation for most users that the search page will remember the last search criteria used. PDSA Framework’s user filter state approach provides the ability to save and restore users’ previous selections and entries to a set of controls in a page, user control or panel in a generalized way so that you do not have to write custom code to re-load a user’s search criteria.

Overview of User Filter State

In this context, a “filter” is a set of values used in a search feature to limit the search results. We refer to the approach of saving a set of search values as “user filter state” because the last set of values used on a search page is the *state* of that *filter*, and these values are stored separately for each user who visits or uses each feature of the application.

This feature set has been designed and tested for use with PDSA Framework web applications, and the class that allows you to save and restore filter state are part of the PDSA.Web namespace.

The PDSAUserFilterState Class

As mentioned above, this is the class you will use add user filter state handling capability to your web page or user control.

Public Methods

There are two public instance methods of this class, Save and Restore, which are used to save and restore the state of values and selections in a control container such as a page, user control or panel.

Values are stored in and restored from the pdsaUserFilterState table as a “state graph”, a representation of the state of the values of each control as an XML string.

1. Save method
    a. Provides the ability to save the state of controls contained in a Control container as an XML state graph.
    b. Parameters:
        i. container – an instance of a Control (System.Web.UI.Control) that has a collection of controls that will be serialized in a state graph.
        ii. formName – a unique identifier of the page, user control or panel. Since a page may have multiple areas you may wish to serialize or restore, make sure that no two calls to the save method use the same identifier.
        iii. excludeControls – a comma-separated list of the controls NOT to serialize in the state graph.
        iv. defaultState – an XML state graph to store in the database as a default. Passing a value for default state overrides the current state and does not raise any of the events associated with serializing the controls.
2. Restore method
    a. Provides the ability to restore the state of controls contained in a Control container from an XML state graph.
    b. Parameters:
        i. container – an instance of a Control (System.Web.UI.Control) that has a collection of controls that will be restored from the serialized a state graph.
        ii. filterSetName – a unique identifier of the page, user control or panel to be restored. Since a page may have multiple areas you may wish to serialize or restore, make sure that no two calls to the restore method use the same identifier, but you must use the same identifier used in the Save() method for the same set of controls.
        iii. excludeControls – a comma-separated list of the controls NOT to restore from the serialized state graph.
        iv. restoreDefault – directs the class to find and restore a state graph not associated with a particular user (where the user identity key is “-1”).

Overridable Methods

The following methods, declared as virtual or overridable, may be overridden and used in an inherited class to add your own additional processing capabilities to the PDSAUserFilterState class.

1. OnAfterRestore
    a. Called from the restore method after a value has been restored to a known control type.
    b. Raises the AfterRestore event.
    c. Parameters identify the identity of the control and the value of the control being restored.
2. OnAfterSave
    a. Called from the save method after a value has been saved into the state graph from a known control type.
    b. Raises the AfterSave event.
    c. Parameters identify the identity of the control and the value of the control being saved.
3. OnBeforeRestore
    a. Called from the restore method before a value is restored to a known control type.
    b. Raises the BeforeRestore event.
    c. Parameters identify the identity of the control and the value of the control being restored.
4. OnBeforeSave
    a. Called from the save method before a value is saved into the state graph from a known control type.
    b. Raises the BeforeSave event.
    c. Parameters identify the identity of the control and the value of the control being saved.
5. OnRestoreGridState
    a. Called from the restore method when a grid control is detected in the state graph.
    b. Raises the RestoreGridState event.
    c. Parameters identify the identity of the grid control, the control value, the page number the grid was on and the number of rows displayed in the grid.
6. OnRestoreUnknownControlType
    a. Called from the restore method when an unknown control type is detected in the state graph.

   b. Raises the RestoreUnknownControlType event.
   c. Parameters identify the identity of the control and the value of the control being restored.

7. OnSaveUnknownControlType
   a. Called from the save method before a value is saved into the state graph for an unknown control type.
   b. Raises the SaveUnknownControlType event.
   c. Parameters identify the identity of the control and the XML text writer instance being used to add XML to the state graph.

Events

The following events are raised during the save and restore operations to give you the opportunity to customize the behavior of the state that is being saved or restored.

Except as noted, each event passes an instance of PDSAUserFilterStateEventArgs which contains the identity of the control being saved or restored and the value of the control.

Note that setting the 'Cancel' property to 'true' will allow you to specify that the operation should not be completed for the current control only for the SaveUnknownControlType event. The 'Cancel' value has no effect on the other events, which are designed to give you an opportunity to inspect the value being saved or restored, or to override the state after it has been restored or saved.

1. AfterRestoreState
   a. Raised after a value has been restored to a known control type.
2. AfterSaveState
   a. Raised after a value has been saved for a known control type.
3. BeforeRestoreState
   a. Raised before a value has been restored to a known control type.
4. BeforeSaveState
   a. Raised before a value has been saved for a known control type.
5. RestoreGridState
   a. Raised when a grid control is encountered in the state graph.
   b. Passes an instance of the PDSAUserGridStateEventArgs class, which contains the item count and last page index of the grid state.
6. RestoreUnknownControlType
   a. Raised when a unknown control is encountered in the state graph.
7. SaveUnknownControlType

a. Raised when an unknown control type is encountered in the container.

Filter State Pre-Requisites

In order to use filter state handling, all of the following elements must be present in your application:

1. References to PDSA Framework libraries, which must include PDSA.Web.dll and PDSA.DataLayer.dll.
2. PDSA Framework tables and stored procedures must exist in the database.
3. At least one application, entity and user must exist in the PDSA Framework database tables.
4. PDSA Framework configuration sections for data providers along with a connection string to the database where the PDSA tables are located.

Adding Filter State

The following steps illustrate how to add filter state handling to a page or user control. In this example, the page or user control contains an ASP.NET Panel control named “pnlFilters” in which a text box, a check box and a drop-down list are located, as shown in Figure 1.

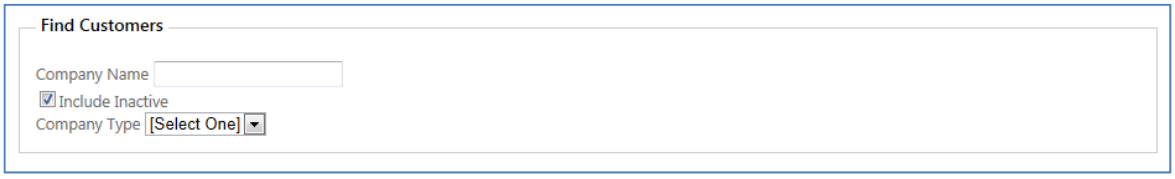

Figure 1, Sample filter panel

1. Make sure that the session variables for application identity, entity identity and current user identity are being set in WebUserSession.

   Note: In a PDSA Framework website created using PDSA Application Builder, this is handled for you in Global.asax and PDSALogin.aspx.

2. Add a private member variable to the code-behind to reference an instance of PDSAUserFilterState.

```
private PDSAUserFilterState _filterState;
```

3. In the OnInit method or Init event handler, create a new instance of PDSAUserFilterState.

```
protected override void OnInit(EventArgs e)
{
   base.OnInit(e);

   _filterState = new PDSAUserFilterState();
}
```

4. Add a call to the Restore method in the OnLoad method or Load event handler.

```
protected void Page_Load(object sender, EventArgs e)
{
  // when the page loads for the first time,
  // restore the previous state

  if (!this.IsPostBack)
  {
    _filterState.Restore(pnlFilters,
      "frmUserFilterState", string.Empty, false);
  }
}
```

5. Add a call to the Save method in the OnPreRender method or PreRender event handler.

```
protected override void OnPreRender(EventArgs e)
{
   base.OnPreRender(e);

   // each time the page returns to the user,
   // save the current state

   _filterState.Save(pnlFilters, "frmUserFilterState",
      string.Empty, string.Empty);
}
```

6. When the page or user control runs, the following XML is stored in the pdsaUserFilterState table (formatted for clarity):

```
<?xml version="1.0" encoding="utf-16"?>
<frmUserFilterState>
  <Parameters>
    <txtCompanyName />
    <chkIncludeInactive>Y</chkIncludeInactive>
    <ddlOptions />
  </Parameters>
</frmUserFilterState>
```

Filter State Sample

A complete working sample of the PDSAUserFilterState class may be found on page “frmUserFilterState.aspx” in the PDSA Web Library Sample solution located under:

[InstallFolder]\Samples\WebForms\PDSAWebLibrarySample

How-To’s

Create a Default State

Use the following steps to create a default state record for a filter panel.

1. Hook up your page or user control as described above in the section “Adding Filter State”.
2. Run the application and navigate to the feature, make sure that a new record is saved to the pdsaUserFilterState table representing the initial state of your controls.
3. Using SQL Server Management Studio, execute the following SQL statement.
    a. Make sure you replace the tokens “<PK OF YOUR NEW RECORD>” and “<YOUR FORM NAME>” in the script below with your values.

```
SET IDENTITY_INSERT PDSA.pdsaUserFilterState ON

INSERT INTO PDSA.pdsaUserFilterState
  (UserFilterStateId, ApplicationId, UserId, EntityId, FormName,
   FilterState, IsDefault, InsertName, InsertDate, UpdateName,
   UpdateDate, ConcurrencyValue)
  SELECT
    CASE
      WHEN
        (SELECT MIN(UserFilterStateId)
           FROM PDSA.pdsaUserFilterState) >= 0
      THEN -1
      ELSE
        (SELECT MIN(UserFilterStateId)
           FROM PDSA.pdsaUserFilterState) - 1
    END AS UserFilterStateId,
    ApplicationId, NULL, EntityId, FormName, FilterState, 1,
    InsertName, InsertDate, UpdateName, UpdateDate,
    ConcurrencyValue
    FROM PDSA.pdsaUserFilterState
    WHERE UserFilterStateId = <PK OF YOUR NEW RECORD>
      AND FormName = '<YOUR FORM NAME>'

SET IDENTITY_INSERT PDSA.pdsaUserFilterState OFF
```

Note: Setting the primary key to a negative number is not required, but it does make identifying and copying a set of defaults from one environment to another faster and easier.

4. If necessary, modify the XML stored to represent the default state. For example, the XML below has been modified to set the check box to false and the company name text box to “PDSA”

```
<?xml version="1.0" encoding="utf-16"?>
<frmUserFilterState>
  <Parameters>
    <txtCompanyName>PDSA<txtCompanyName>
    <chkIncludeInactive>N</chkIncludeInactive>
    <ddlOptions />
  </Parameters>
</frmUserFilterState>
```

5. Add a button, image button or link button to your page or user control to provide users with the ability to restore the default state.
6. In the code-behind of your page or user control, handle the click even for the new control by adding a call to Restore() in which you pass “True” for the restoreDefault parameter value:

```
protected void btnRestoreDefault_Click(object sender, EventArgs e)
{
   _filterState.Restore(pnlFilters, "frmUserFilterState",
      string.Empty, true);
}
```

7. Following these steps with the sample in the PDSA Web Library Sample solution results in the display shown in Figure 2, below.

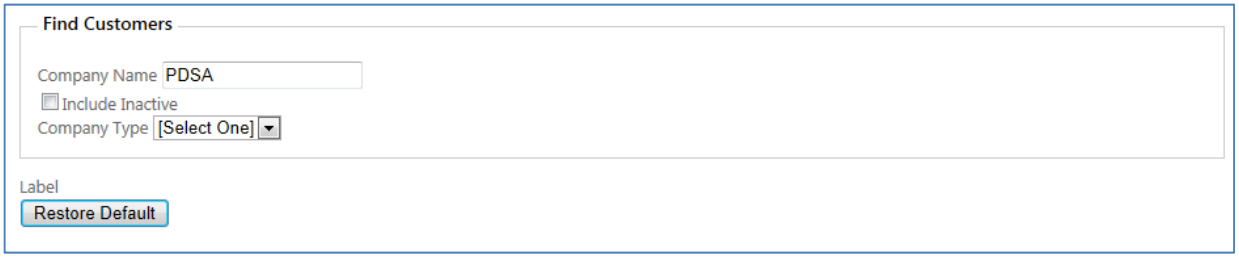

Figure 2, Filter panel with default state restored.

Customize State Handling

Using the filter state class, you are able to customize the way state is saved or restored to your filter panel. As documented above in this chapter, there are six events raised by the filter state class before and after saving or restoring state and in response to unknown types of controls.

For example, you may have a business rule that the "include inactive" check box is always selected, no matter what the user's previous selection. Or you might verify that a drop-down list has been loaded with values before state is restored.

1. First, bind event handlers to each of the events you wish to handle in the Init event handler or OnInit override.
    a. In the example below, you will see the complete set of event handlers.

```
protected override void OnInit(EventArgs e)
{
   base.OnInit(e);

  _filterState = new PDSAUserFilterState();

  _filterState.BeforeSaveState +=
    new PDSAFilterStateEventHandler(BeforeSaveState);
  _filterState.AfterSaveState +=
    new PDSAFilterStateEventHandler(AfterSaveState);
  _filterState.BeforeRestoreState +=
    new PDSAFilterStateEventHandler(BeforeRestoreState);
  _filterState.AfterRestoreState +=
    new PDSAFilterStateEventHandler(AfterRestoreState);
  _filterState.SaveUnknownControlType +=
    new PDSAFilterStateEventHandler(SaveUnknownControlType);
  _filterState.RestoreUnknownControlType +=
    new PDSAFilterStateEventHandler(RestoreUnknownControlType);
}
```

2. Add logic to one or more event handlers to override the default behavior.

```
void _filterState_AfterRestoreState(object sender,
       PDSAUserFilterStateEventArgs e)
{
  if (e.ControlId == chkIncludeInactive.ID)
  {
    chkIncludeInactive.Checked = true;
  }
}
```

3. When running the page, you will see that the check box is restored to checked even if the last user selection was ‘unchecked’.

Handle an Unknown Control

Since the only controls handled by default are the text box, check box and drop-down list, you must handle saving and restoring values for all other custom or third-party controls.

In order to handle an unknown control, follow these steps:

1. Make sure you bind event handlers to each of the events you wish to handle in the Init event handler or OnInit override.
    a. In the example below, you will see the complete set of event handlers.

```
protected override void OnInit(EventArgs e)
{
   base.OnInit(e);

  _filterState = new PDSAUserFilterState();

  _filterState.BeforeSaveState +=
    new PDSAFilterStateEventHandler(BeforeSaveState);
  _filterState.AfterSaveState +=
    new PDSAFilterStateEventHandler(AfterSaveState);
  _filterState.BeforeRestoreState +=
    new PDSAFilterStateEventHandler(BeforeRestoreState);
  _filterState.AfterRestoreState +=
    new PDSAFilterStateEventHandler(AfterRestoreState);
  _filterState.SaveUnknownControlType +=
    new PDSAFilterStateEventHandler(SaveUnknownControlType);
  _filterState.RestoreUnknownControlType +=
    new PDSAFilterStateEventHandler(RestoreUnknownControlType);
}
```

2. Add custom logic to the SaveUnknownControlType event handler.
    a. In this example, the event handler detects the identity of a hidden field and stores its value in the event argument 'Value' property.

```
void _filterState_SaveUnknownControlType(object sender,
PDSAUserFilterStateEventArgs e)
{
   if (e.ControlId == "mHiddenField")
   {
      e.Value = mHiddenField.Value;
   }
   else
   {
      e.Cancel = true;
   }
}
```

3. Add custom logic to the RestoreUnknownControlType event handler.
    a. In this example, the event handler detects the identity of an unknown control in the state graph and sets its value.

```
void _filterState_RestoreUnknownControlType(object sender,
PDSAUserFilterStateEventArgs e)
{
   if (e.ControlId == "mHiddenField")
   {
      mHiddenField.Value = e.Value;
   }
}
```

4. When running the page, you will see that the hidden control value is saved and restored.

Summary

In this chapter, you have learned all the features of filter state handling in a PDSA Framework web application and step-by-step instructions for how to add filter state handling to your PDSA Framework web application.