Sun GlassFish Web Space Server 10.0 Developer's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–7050
February 2009

090728@22510

Contents

◆ ◆ ◆ CHAPTER 1

Development Tasks

This chapter explains how you can use certain plug-ins available as part of Sun GlassFish Web Space Server 10.0 to develop portlets.

Tip – Some deep dive examples for Sun GlassFish Web Space Server are available at `http://wikis.sun.com/display/websynergy/Java+Developer+Technology+-+Web+Space+Deep+Dive+Examples`.

This chapter includes the following sections:

Developing Portlets and Themes Using NetBeans Portal Pack 3.0 for Sun GlassFish Web Space Server

The Portal Pack 3.0 provides a set of plug-ins for NetBeans™ IDE to support the portlet application development life-cycle of within NetBeans IDE. Using these set of plug-ins, developers can develop, package, deploy, and test portlets within the NetBeans IDE. By automatic code and Deployment Descriptor generation, the Portal Pack helps developers to develop portlets quickly. Portal Pack plug-ins provide a tight integration with portal servers

such as the Sun GlassFish Web Space Server 10.0, Liferay Portal Server, Sun Java System Portal Server 7.x, and OpenPortal Portlet Container on `java.net` to support deployment and undeployment of portlets on both the local and remote servers.

For more information on how to develop portlets using Portal Pack 3.0 on NetBeans IDE, see Using Portal Pack 3.0 Plug-ins to Create Portlets on NetBeans 6.5 IDE.

NetBeans Portal Pack also provides as alternative to ViewDesigner plug-ins for developing themes. For the information on developing themes using Portal Pack plug-ins, see `http://www.netbeans.org/kb/65/web/portal-pack-themes.html` or `http://www.netbeans.org/kb/docs/web/portal-pack-themes.html`.

For the information on developing themes using ViewDesigner plug-ins, see "Developing Themes Using the ViewDesigner Plug-in in Sun GlassFish Web Space Server" on page 37.

Developing Services Using the Service Builder Plug-in in NetBeans Portal Pack 3.0

The Service Builder plug-in is part of the NetBeans Portal Pack 3.0 release. The Service Builder plug-in in the Sun GlassFish Web Space Server provides a GUI editor to automate the creation of interfaces and classes that are used by a given portal or a portlet.

This section explains how you can use the Service Builder plug-in to automatically create the interfaces and classes that can be used by a portlet. To do this, you need to create a portlet and deploy it on Sun GlassFish Web Space Server.

Creating a Portlet Using NetBeans Portal Pack 3.0

Create a portlet called MovieDatabase using the NetBeans IDE.

▼ To Create a Portlet Using the NetBeans IDE

1. **On NetBeans IDE, click File and select New Project.**
   The New Project window appears.
2. **From the list of Projects in the Choose Project pane, select Web Application and click Next.**
3. **In the New Web Application window, specify the name of the project as MovieApplication.**
4. **Click Next.**
5. **In the Server and Settings pane, select Sun GlassFish Web Space Server from the Server drop down menu.**

6. **Click Next.**

7. **From the Frameworks list box, select Portlet Support framework.**

8. **In the Portlet Support Configuration pane, select Create Portlet and change the Portlet Class Name to MovieDataEntryPortlet.**

9. **Click Finish.**

   The NetBeans IDE creates a portlet web application and generates all the required files for the web application.

Creating the `service.xml` File

After creating a portlet application, you need to create a `service.xml` file inside the portlet application.

▼ To Create a `service.xml` File

1. **From the MovieApplication portlet, right click Web Pages and select New.**

2. **Click Other.**

   The New File wizard appears.

3. **In the Choose File Type pane, choose MovieApp from the Project drop down menu.**

4. **Select Sun GlassFish Web Space Server Portlets from Categories and Service Builder XML from File Types.**

5. **Click Next.**

   The Name and Location window appears.

6. **Specify the File Name as service and click Finish.**

   The `service.xml` file is created inside the `web` directory. The Design tab is created where you can see the GUI editor for the `service.xml`.

▼ Creating Entities in the `service.xml` GUI Editor

1. **Click Add.**

   The Service Definition window appears.

2. **Specify the Entity Name as Movie and the Table Name for the entity as tb_movie.**

3. **Select the Remote Service checkbox and click Add.**

   The entity Movie is created with the Remote value as true.

4. **Change the default Package Path name to com.sample.movie.**

   After creating an entity, you need to add columns for the entity.

5. **Click Add under the Columns tab.**

   The Column Details window appears.

6. **Specify the Name of the column as name and the column type as String.**

   You can choose the column type as String, Double, and Date from the drop down list.

7. **Select the Primary Key checkbox and click Add.**

   A column is created with the Column Name as name and type as String.

8. **Add three more columns with the Column Types cast, story, and releaseDate with types String, String, and Date respectively.**

   You can see that there are four columns created for the Movie entity.

9. **Add another entity called MovieFeedback with the Table Name as tb_moviefeedback.**

10. **Add columns such as id, movieName, rating, and feedback with types long, string, string, and string respectively.**

    You can see that there are four columns created for the MovieFeedback entity.

    You need to add a Finder method for the MovieFeedback entity.

11. **Click the Finders tab.**

12. **Click Add Finder.**

    The Finder Details window appears.

13. **Specify the Finder Name as MovieName.**

14. **From the Available Columns pane, select movieName and click Add.**

    The `movieName` method is added under the Finder Columns pane. Since the `movieName` is the finder method, you can search the details of a movie by the name of a movie.

    If you want to set the properties for the finder method, click Properties and choose any Comparators.

15 **Click Add.**

You can see that the finder method is added to the `service.xml` and the same method appears in GUI editor.

Generating Services

1 **From the `service.xml` GUI editor, click the Generate Services tab.**

You can see the creation of classes, services, and data models that are required for database interaction.

2 **In the GUI editor, click the Local Methods tab and click Go To Source.**

The local service file for the Movie entity appears.

3 **In the local service class file, add a method called `getMovies` to return the movies listed in the Movie entity.**

You can use the `moviePersistance.findAll` method to find the movies present in the Movie entity.

4 **Save the class file.**

In the GUI editor, you can see the `getMovies` method under the Local Methods tab.

5 **Click Generate Services.**

You can see that all the services are generated again.

Creating the View Page for the Movie Data Entry Portlet

1 **From the Projects pane, click the `MovieDataEntryPortlet_view.jsp` file.**

2 **Add code to the `MovieDataEntryPortlet_view.jsp` file to create a GUI form to specify the movie name, cast, and release date.**

This information is stored in the Movie entity.

3 **From the Projects pane, click the `MovieDataEntryPortlet.java` file under the `com.test` folder.**

The `MovieDataEntryPortlet.java` file appears in the right pane. You can see all the parameters to display the movie details are listed in this file. You can also add other required parameters in this file.

If you expand the folders in the Project pane, you can see the classes that are automatically generated by the service builder infrastructure. If you click the Files tab, you can see all the `.jar` and `.sql` files that are automatically created. When you deploy the portlet on Sun GlassFish Web Space Server 10.0 or Liferay portal server, the .sql files are called and the tables will be created automatically.

Creating the MovieList Portlet

1. **Right click on the MovieApp portlet, select New and click Other.**
   The New File window appears.
2. **Select Portlet from Categories and Portlet from File Types.**
3. **Click Next.**
   The New Portlet window appears.
4. **In the Name and Location pane, specify the Class Name as MovieList, Package as com.test, and Portlet Name as MovieList.**
5. **Click Next.**
6. **In the Create Jsps for Portlet pane, specify the View Jsp as** `MovieList_view.jsp`**.**
7. **Click Finish.**
   The MovieList portlet is created. This portlet lists the movies in the Movie portlet database or Movie entity.
8. **From the Projects pane, click the** `MovieList_view.jsp` **file.**
   The `MovieList_view.jsp` file appears in the right pane.
9. **Add code to the** `MovieList_view.jsp` **file to create a GUI for the MovieList portlet.**
   Using this JSP file, you can fetch all the data from Movie entity from the Movie services class and display the movie details in the GUI form. In this JSP file, you can use the `getMovies` method to fetch the movie data.
10. **From the Projects pane, right click on the MovieApp portlet and click Run to deploy the portlet.**
    The Sun GlassFish Web Space Server server is started. The MovieDataEntry and MovieList portlets are deployed successfully.
11. **In the Sun GlassFish Web Space Server 10.0, login as Admin user.**
12. **Add the MovieDataEntry and MovieList portlets in a portal page.**

Adding Portlets in Sun GlassFish Web Space Server

1. **From the Welcome menu, click Add Application.**
   The Add Application window appears. You can see the MovieDataEntry and MovieList portlets under User_Portlets.

2 **Click Add to add the MovieDataEntry and MovieList portlets in the portal page.**

3 **In the MovieDataEntry portlet, add new records by specifying movie name, cast, story, and release date.**

You can see that the movie records that you have entered in the MovieDataEntry portlet appears in the MovieList portlet.

Defining the Database Structure

You need to define the structure of a database to expose your services to a local or remote service. You can define the structure of the database using the Service Builder plug-in user interface. After you define the structure, you can generate the `service.xml` file using the Generate service button.

For example, if you want to create a movie database, define the required elements in the Service Builder plug-in. Using the defined elements, the Service Builder plug-in automatically creates the `service.xml` file with the defined values.

Structure of `service.xml` File

The `service.xml` file is available in the `/docroot/WEB-INF` folder. A typical `service.xml` file looks like this.

```
<?xml version="1.0"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 5.0.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_5_0_0.dtd">

<service-builder package-path="com.sample.portlet.library">
        <namespace>Library</namespace>
        <entity name="Book" local-service="true" remote-service="false">

            <!-- PK fields -->

            <column name="bookId" type="long" primary="true" />

            <!-- Group Instance -->

            <column name="groupId" type="long" />

            <!-- Audit fields -->

            <column name="companyId" type="long" />
            <column name="userId" type="long" />
            <column name="userName" type="String" />
```

```
            <column name="createDate" type="Date" />
            <column name="modifiedDate" type="Date" />

            <!.. Other Fields ..>

            <column name="title" type="String" />
        </entity>
</service-builder>
```

Let us look at the elements and their functions in the `service.xml` file.

```
<service-builder package-path="com.sample.portlet.library">
```

The package-path element specifies the package path that the class will generate to. In this example, classes will generate to `WEB-INF/src/com/sample/portlet/library/`.

```
<namespace>Library</namespace>
```

The namespace element must be a unique namespace for this component. The names of tables names will be prepended with this namespace element.

```
<entity name="Book" local-service="true" remote-service="false">
```

The entity name is the database table you want to create.

Generating a Service Layer

Open up a command prompt, navigate to your portlet's root folder and type the following command.

```
ant build-service
```

The service layer has been generated successfully when you see BUILD SUCCESSFUL message.

Creating the Service Layer Class

Open the `BookLocalServiceImpl.java` file, which is available in `/docroot/WEB-INF/src/com/sample/portlet/library/service/impl/`. We will be adding the database interaction methods to this service layer class. Add the following method to the `BookLocalServiceImpl` class.

```
public Book addBook(long userId, String title)
            throws PortalException, SystemException {
        User user = UserUtil.findByPrimaryKey(userId);
        Date now = new Date();
```

```
        long bookId = CounterLocalServiceUtil.increment(Book.class.getName());

        Book book = bookPersistence.create(bookId);

        book.setTitle(title);
        book.setCompanyId(user.getCompanyId());
        book.setUserId(user.getUserId());
        book.setUserName(user.getFullName());
        book.setCreateDate(now);
        book.setModifiedDate(now);
        book.setTitle(title);
        return bookPersistence.update(book);
}
```

Before you can use the code, you must regenerate the service layer class. Navigate to the root folder of your portlet, and type the `ant build-service` command. You can now add a new book to the database by making the following call.

```
BookLocalServiceUtil.addBook(userId, ?A new title?);
```

Security and Permissions Service

The Sun GlassFish Web Space Server implements a fine-grained permissions system, which developers can use to implement access security into their custom portlets, giving administrators and users a lot more control over their portlets and content. This section of the document will provide a reference for implementing security into custom portlets.

Overview

Adding fine-grained permissions to custom portlets consists of four main steps. This procedure is also known as DRAC.

1. Define all resources and their permissions.
2. Register all the resources mentioned in step 1 into the permissions system. This is also known simply as “adding resources”.
3. Associate the necessary permissions to these resources.
4. Check permission before returning resources.

Implementing Permissions

In this section, each of the four main steps in adding the security feature available in the Sun GlassFish Web Space Server into custom portlets (built on top of the Sun GlassFish Web Space Server portal) will be explained. The following are two definitions that are important to remember.

- Resource — A generic term for any object represented in the portal. Examples of resources include portlets (Message Boards, Calendar, and so on), Java classes (Message Board Topics, Calendar Events, and so on), and files (documents, images, and so on).
- Permission — An action acting on a resource. For example, the view in "viewing the calendar portlet" is defined as a permission in the Sun GlassFish Web Space Server.

Note – The permission for a portlet resource is implemented a little differently from the other resources such as Java classes and files. In each of the subsections below, the permission implementation for the portlet resource is explained first, then the model (and file) resource.

For your custom portlet, the Sun GlassFish Web Space Server portal needs to know whether there are resources that require permission and whether there are custom permissions. The default configuration is encapsulated in an XML file found in the portal source under the `/portal-impl/classes/resource-actions` directory; you might use it as a reference to create a similar file for your portlet. There is also a Sample Permissions portlet available in the Plugins project on SourceForge. If your portlet only needs the view and the configuration permission, and that the portlet does not use any models with permission, then you do not need to create this XML file. The reason is that all portlets in the Sun GlassFish Web Space Server automatically inherit these permissions. However, if your portlet does have custom permission and/or uses models that have custom permissions, then you will need to create an XML file defining the resources and actions. Let us take a look at `blogs.xml` file in `portal/portal-impl/classes/resource-actions` and see how the blogs portlet defined these resources and actions:

```
<?xml version="1.0"?>
<resource-action-mapping>
        <portlet-resource>
        <portlet-name>33</portlet-name>
        <supports>
            <action-key>ADD_ENTRY</action-key>
            <action-key>CONFIGURATION</action-key>
            <action-key>VIEW</action-key>
        </supports>
        <community-defaults>
            <action-key>VIEW</action-key>
        </community-defaults>
        <guest-defaults>
            <action-key>VIEW</action-key>
```

```
            </guest-defaults>
            <guest-unsupported>
                <action-key>ADD_ENTRY</action-key>
            </guest-unsupported>
        </portlet-resource>
        <model-resource>
            <model-name>com.liferay.portlet.blogs.model.BlogsEntry</model-name>
            <portelt-ref>
                <portlet-name>33</portlet-name>
            </portelt-ref>
            <supports>
                <action-key>ADD_DISCUSSION</action-key>
                <action-key>DELETE</action-key>
                <action-key>DELETE_DISCUSSION</action-key>
                <action-key>>PERMISSIONS</action-key>
                <action-key>UPDATE</action-key>
                <action-key>UPDATE_DISCUSSION</action-key>
                <action-key>VIEW</action-key>
            </supports>
            <community-defaults>
                <action-key>ADD_DISCUSSION</action-key>
                <action-key>VIEW</action-key>
            </community-defaults>
            <guest-defaults>
                <action-key>VIEW</action-key>
            </guest-defaults>
            <guest-unsupported>
                <action-key>ADD_DISCUSSION</action-key>
                <action-key>DELETE</action-key>
                <action-key>DELETE_DISCUSSION</action-key>
                <action-key>>PERMISSIONS</action-key>
                <action-key>UPDATE</action-key>
                <action-key>UPDATE_DISCUSSION</action-key>
            </guest-unsupported>
        </model-resource>
    </resource-action-mapping>
```

In the XML, the first thing defined is the portlet itself. Right under the root element <resource-action-mapping>, we have a child element called <portlet-resource>. In this element, we define the portlet name, which is 33 in our case. Next, we list all the actions this portlet supports under the <supports> tag. Keep in mind that this is at the portlet level. To understand what should be listed here, developers should ask themselves what actions belong to the portlet itself or what actions are performed on the portlet that may require a security check. In our case, users need permission to add an entry (ADD_ENTRY), configure blogs portlet settings (CONFIGURATION), and view the blogs itself (VIEW). Each of these supported permissions is within its own <action-key> tag. After we?ve defined all the actions that require a check, we move on to define some of the default permission settings. The community-defaults tag defines what actions are permitted by default for this

portlet on the community (group) page upon which the portlet resides. To put it another way, what should a user that has access to the community in which this portlet resides be able to do minimally. For the blogs portlet, a user with access to the community containing the blogs portlet should be able to view it. Likewise, the guest-defaults tag defines what actions are permitted by default to guests visiting a layout containing this portlet. So if a guest has access to the community page that contains a blogs portlet, the guest should, at the very least, be able to view the portlet according to `blogs.xml` (not necessarily the content of the portlet). Otherwise, the guest will see an error message within the portlet. Depending on your custom portlet, you may add more actions here that make sense. The <guest-unsupported> tag contains actions that a visiting guest should never be able to do. For example, the guest visiting the blogs portlet should never be able to add a blog entry since the blog belongs to either a user or a group of users. So even if a user wants to grant guests the ability to add a blog entry to her blog, there is no way for her to grant that permission because the `blogs.xml` does not permit such an action for guests.

After defining the portlet as a resource, we move on to define models within the portlet that also require access check. The model resource is surrounded by the <model-resource> tag. Within this tag, we first define the model name. This must be the fully qualified Java class name of the model. Next we define the portlet name that this model belongs to under the portlet-ref tag. Though unlikely, a model can belong to multiple portlets, which you may use multiple <portlet-name> tags to define. Similar to the portlet resource element, the model resource element also allows you to define a supported list of actions that require permission to perform. You must list out all the performable actions that require a permission check. As you can see for a blog entry, a user must have permission in order to add comments to an entry, delete an entry, change the permission setting of an entry, update an entry, or simply to view an entry. The <community-defaults> tag, and the <guest-unsupported> tag are all similar in meaning to what?s explained above for a portlet resource.

After defining your permission scheme for your custom portlet, you then need to inform the Sun GlassFish Web Space Server the location of this file. For the Sun GlassFish Web Space Server core, the XML file would normally reside in `portal/portalimpl/classes/resource-actions` and a reference to the file would appear in the `default.xml` file. For a plugin, you should put the file in a directory that is in the class path for the project. Then create a properties file for your portlet (the one in the Sample Permissions Portlet is simply called, `sample-permissionsportlet. properties`) and create a property called `resource.actions.configs` with a value that points to the the XML file. Below is an example from the Sample Permissions Portlet:

```
resource.actions.configs=resource-actions/sample-permissions-portlet.xml
```

- Adding Resource — After defining resources and actions, the next task is to write code that adds resources into the permissions system. A lot of the logic to add resources is encapsulated in the `ResourceLocalServiceImpl` class. So adding resources is as easy as calling the add resource method in `ResourceLocalServiceUtil` class.

```
public void addResources(
        String companyId, String groupId, String userId, String name,
        String primKey, boolean portletActions,
        boolean addCommunityPermissions, boolean addGuestPermissions);
```

For all the Java objects that require access permission, you need to make sure that they are added as resources every time a new one is created. For example, every time a user adds a new entry to her blog, the `addResources(...)` method is called to add the new entry to the resource system. Here is an example of the call from the `BlogsEntryLocalServiceImpl` class.

```
ResourceLocalServiceUtil.addResources(
        entry.getCompanyId(), entry.getGroupId(), entry.getUserId(),
        BlogsEntry.class.getName(), entry.getPrimaryKey().toString(),
        false, addCommunityPermissions, addGuestPermissions);
```

The parameters companyId, groupId, and userId should be self explanatory. The name parameter is the fully qualified Java class name for the resource object being added. The *primKey* parameter is the primary key of the resource object. As for the *portletActions* parameter, set this to true if you are adding portlet action permissions. In our example, we set it to false because we are adding a model resource, which should be associated with permissions related to the model action defined in `blogs.xml`. The *addCommunityPermissions* and the *addGuestPermissions* parameters are inputs from the user. If set to true, `ResourceLocalService` will then add the default permissions to the current community group and the guest group for this resource respectively.

If you would like to provide your user the ability to choose whether to add the default community permission and the guest permission for the resources within your custom portlet, Liferay has a custom JSP tag you may use to quickly add that functionality. Simply insert the <liferay-ui:input-permissions /> tag into the appropriate JSP and the checkboxes will show up on your JSP. Of course, make sure the tag is within the appropriate <form> tags.

To prevent having a lot of dead resources taking up space in the Resource_ database table, you must remember to remove them from the Resource_ table when the resource is no longer applicable. Simply call the `deleteResource(...)` method in `ResourceLocalServiceUtil`. Here is an example of a blogs entry being removed:

```
ResourceLocalServiceUtil.deleteResource(
        entry.getCompanyId(), BlogsEntry.class.getName(),
        Resource.TYPE_CLASS, Resource.SCOPE_INDIVIDUAL,
        entry.getPrimaryKey().toString());
```

- Adding Permission — On the portlet level, no code needs to be written in order to have the permission system work for your custom portlet. Your custom portlet will automatically have all the permission features. If you have defined any custom permissions (supported actions) in your portlet-resource tag in section 3.1, those are automatically added to a list of

permissions and users can readily choose them. Of course, for your custom permissions to have any value, you will need to show or hide certain functionality in your portlet. You can do that by checking the permission first before performing the intended functionality.

In order to allow a user to set permissions on the model resources, you will need to expose the permission interface to the user. This can be done by adding two Liferay UI tags to your JSP. The first one is the <liferay-security:permissionsURL> tag which returns a URL that takes the user to the page to configure the permission settings. The second tag is the <liferay-ui:icon> tag that shows a permission icon to the user. Below is an example found in the `view_entry_content.jspf` file.

```
<liferay-security:permissionsURL
        modelResource="<%= BlogsEntry.class.getName() %>"
        modelResourceDescription="<%= entry.getTitle() %>"
        resourcePrimKey="<%= entry.getPrimaryKey().toString() %>"
        var="entryURL"
/>

<liferay-ui:icon image="permissions" url="<%= entryURL %>" />
```

The attributes you need to provide to the first tag are modelResource, modelResourceDescription, resourcePrimKey, and var. The modelResource attribute is the fully qualified Java object class name. It then gets translated in `Language.properties` to a more readable name.

As for the modelResourceDescription attribute, you can pass in anything that best describes this model instance. In the example, the blogs title was passed in. The resourcePrimKey attribute is simply the primary key of your model instance. The var attribute is the variable name this URL String will get assigned to. This variable is then passed to the <liferay-ui:icon> tag so the permission icon will have the proper URL link. There is also an optional attribute redirect that is available if you want to override the default behavior of the upper right arrow link. That is all you need to do to enable users to configure the permission settings for model resources.

- Checking Permissions — The last major step to implementing permission to your custom portlet is to check permission. This may be done in a couple of places. For example, your business layer should check for permission before deleting a resource, or your user interface should hide a button that adds a model (For example, a calendar event) if the user does not have permission to do so.

  Similar to the other steps, the default permissions for the portlet resources are automatically checked for you. You do not need to implement anything for your portlet to discriminate whether a user is allowed to view or to configure the portlet itself. However, you do need to implement any custom permission you have defined in your `resource-actions` XML file. In the blogs portlet example, one custom supported action is ADD_ENTRY. There are two places in the source code that check for this permission. The first one is in the `view_entries.jsp` file. The presence of the add entry button is contingent on whether the user has permission to add entry (and also whether the user is in tab one).

```
<%
boolean showAddEntryButton = tabs1.equals("entries") &&
PortletPermission.contains(permissionChecker, plid, PortletKeys.BLOGS,
ActionKeys.ADD_ENTRY);
%>
```

The second place that checks for the add entry permission is in the `BlogsEntryServiceImpl` file. Notice the difference between this file and the `BlogsEntryLocalServiceImpl`. In the `addEntry(...)` method, a call is made to check whether the incoming request has permission to add entry.

```
PortletPermission.check(
        getPermissionChecker(), plid, PortletKeys.BLOGS,
        ActionKeys.ADD_ENTRY);
```

If the check fails, it throws a `PrincipalException` and the add entry request aborts. You are probably wondering what the `PortletPermission` and the `PermissionChecker` classes do. Let us take a look at these two classes.

The `PermissionChecker` class has a method called `hasPermission(..)` that checks whether a user making a resource request has the necessary access permission. If the user is not signed in (guest user), it checks for guest permissions. Otherwise, it checks for user permissions. This class is available to you in two places. First in your business logic layer, you can obtain an instance of the `PermissionChecker` by calling the `getPermissionChecker()` method inside your `ServiceImpl` class. This method is available because all `ServiceImpl` (not `LocalServiceImpl`) classes extend the `PrincipalBean` class, which implements the `getPermissionChecker()` method. The other place where you can obtain an instance of the `PermissionChecker` class is in your JSP files. If your JSP file contains the portlet tag <portlet:defineObjects /> or includes another JSP file that does, you will have an instance of the `PermissionChecker` class available to you via the `permissionChecker` variable. Now that you know what the `PermissionChecker` does and how to obtain an instance of it, let us take a look at Liferay's convention in using it.

`PortletPermission` is a helper class that makes it easy for you to check permission on portlet resources (as opposed to model resources, covered later). It has two static methods called `check(...)` and another two called `contains(...)`. They are all essentially the same. The two differences between them are:

1. one `check(...)` method and one `contains(...)` method take in the portlet layout ID variable (plid).
2. The `check(...)` methods throw a new `PrincipalException` if user does not have permission, and the `contains(...)` methods return a boolean indicating whether user has permission.

The `contains(...)` methods are meant to be used in your JSP files since they return a boolean instead of throwing an exception. The `check(...)` methods are meant to be called

in your business layer (`ServiceImpl`). Let us revisit the blogs portlet example below. (The `addEntry(...)` method is found in `BlogsEntryServiceImpl`.)

```
public BlogsEntry addEntry(
            long plid, String title, String content, int displayDateMonth,
            int displayDateDay, int displayDateYear, int displayDateHour,
            int displayDateMinute, String[] tagsEntries,
            boolean addCommunityPermissions, boolean addGuestPermissions,
            ThemeDisplay themeDisplay)
        throws PortalException, SystemException {

        PortletPermissionUtil.check(
            getPermissionChecker(), plid, PortletKeys.BLOGS,
            ActionKeys.ADD_ENTRY);

        return blogsEntryLocalService.addEntry(
            getUserId(), plid, title, content, displayDateMonth, displayDateDay,
            displayDateYear, displayDateHour, displayDateMinute, tagsEntries,
            addCommunityPermissions, addGuestPermissions, themeDisplay);
}
```

Before the `addEntry(...)` method calls `BlogsEntryLocalServiceUtil.addEntry(...)` to add a blogs entry, it calls `PortletPermission.check(...)` to validate user permission. If the check fails, a `PrincipalException` is thrown and an entry will not be added. Note the parameters passed into the method. Again, the `getPermissionChecker()` method is readily available in all `ServiceImpl` classes. The plid variable is passed into the method by its caller (most likely from a `PortletAction` class). PortletKeys.BLOGS is just a static String indicating that the permission check is against the blogs portlet. ActionKeys.ADD_ENTRY is also a static String to indicate the action requiring the permission check. You are encouraged to do likewise with your custom portlet names and custom action keys.

Whether you need to pass in a portlet layout ID (plid) depends on whether your custom portlet supports multiple instances. Let us take a look at the message board portlet for example. A community may need three separate page layouts, each having a separate instance of the message board portlet. Only by using the portlet layout ID will the permission system be able to distinguish the three separate instances of the message board portlet. This way, permission can be assigned separately in all three instances. Though in general, most portlets will not need to use the portlet layout ID in relation to the permission system.

Since the `ServiceImpl` class extends the `PrincipalBean` class, it has access to information of the current user making the service request. Therefore, the `ServiceImpl` class is the ideal place in your business layer to check user permission. Liferay's convention is to implement the actual business logic inside the `LocalServiceImpl` methods, and then the `ServiceImpl` calls these methods via the `LocalServiceUtil` class after the permission check completes successfully. Your `PortletAction` classes should make calls to `ServiceUtil` (wrapper to `ServiceImpl`) guaranteeing that permission is first checked before the request is fulfilled.

Checking model resource permission is very similar to checking portlet resource permission. The only major difference is that instead of calling methods found in the `PortletPermission` class mentioned previously, you need to create your own helper class to assist you in checking permission. The next section will detail how this is done.

It is advisable to have a helper class to help check permission on your custom models. This custom permission class is similar to the `PortletPermission` class but is tailored to work with your custom models. While you can implement this class however you like, we encourage you to model your implementation after the `PortletPermission` class, which contains four static methods. Let us take a look at the `BlogsEntryPermission` class.

```
public class BlogsEntryPermission {

        public static void check(
                PermissionChecker permissionChecker, long entryId, String actionId)
            throws PortalException, SystemException {

            if (!contains(permissionChecker, entryId, actionId)) {
                throw new PrincipalException();
            }
        }

        public static void check(
                PermissionChecker permissionChecker, BlogsEntry entry,
                String actionId)
            throws PortalException, SystemException {

            if (!contains(permissionChecker, entry, actionId)) {
                throw new PrincipalException();
            }
        }

        public static boolean contains(
                PermissionChecker permissionChecker, long entryId, String actionId)
            throws PortalException, SystemException {

            BlogsEntry entry = BlogsEntryLocalServiceUtil.getEntry(entryId);

            return contains(permissionChecker, entry, actionId);
        }

        public static boolean contains(
                PermissionChecker permissionChecker, BlogsEntry entry,
                String actionId)
            throws PortalException, SystemException {

            return permissionChecker.hasPermission(
                entry.getGroupId(), BlogsEntry.class.getName(), entry.getEntryId(),
```

```
                actionId);
        }
    }
```

Again, the two `check(...)` methods are meant to be called in your business layer, while the two `contains(...)` methods can be used in your JSP files. As you can see, it is very similar to the `PortletPermission` class. The two notable differences are:

1. Instead of having the *portletId* as one of the parameters, the methods in this custom class take in either an entryId or a BlogsEntry object.
2. None of the methods need to receive the portlet layout ID (plid) as a parameter. (Your custom portlet may choose to use the portlet layout ID if need be.)

Let us see how this class is used in the blogs portlet code.

```
public BlogsEntry getEntry(String entryId) throws PortalException, SystemException {
        BlogsEntryPermission.check(
            getPermissionChecker(), entryId, ActionKeys.VIEW);
        return BlogsEntryLocalServiceUtil.getEntry(entryId);
}
```

In the `BlogsEntryServiceImpl` class is a method called `getEntry(...)`. Before this method returns the blogs entry object, it calls the custom permission helper class to check permission. If this call does not throw an exception, the entry is retrieved and returned to its caller.

```
<c:if test="<%= BlogsEntryPermission.contains(permissionChecker, entry, ActionKeys.UPDATE)
%>">
        <portlet:renderURL windowState="<%= WindowState.MAXIMIZED.toString() %>"
var="entryURL">
            <portlet:param name="struts_action" value="/blogs/edit_entry" />
            <portlet:param name="redirect" value="<%= currentURL %>" />
            <portlet:param name="entryId" value="<%= entry.getEntryId() %>" />
        </portlet:renderURL>

        <liferay-ui:icon image="edit" url="<%= entryURL %>" />
</c:if>
```

In the `view_entry_content.jsp` file, the `BlogsEntryPermission.contains(...)` method is called to check whether or not to show the edit button. That is all there is to it!

Let us review what we have just covered. Implementing permission into your custom portlet consists of four main steps. First step is to define any custom resources and actions. Next step is to implement code to register (or add) any newly created resources such as a BlogsEntry object. The third step is to provide an interface for the user to configure permission. Lastly, implement code to check permission before returning resources or showing custom features. Two major resources are portlets and Java objects. There is not a lot that needs to be done for the portlet resource to implement the permission system since Liferay Portal has a lot of that work done for

you. You mainly focus your efforts on any custom Java objects you have built. You are now well on your way to implement security to your custom Liferay portlets!

Developing Portlets Using Eclipse Portal Pack in Sun GlassFish Web Space Server

The Eclipse Portal Pack provides a set of plug-ins that help you to develop JSR 168/286 Portlets and deploy them on supported portlet containers.

There are three plug-ins available in the Eclipse Portal Pack.

- JSR 168/286 Portlet Builder — This plug-in allows you to create a JSR 168/286 portlet project using the wizards available in the Eclipse IDE. Once you create a portlet project using the wizards, the Eclipse Portal Pack plug-in creates the appropriate class files and updates the `portlet.xml` file. In addition, the plug-in provides a `build.xml` file that allows you to build a WAR, which you can deploy on any portal.
- Sun GlassFish Web Space Server — This plug-in registers the Sun GlassFish Web Space Server that runs on GlassFish server in Eclipse IDE, as a Server Runtime. Currently, the Sun GlassFish Web Space Server plug-in allows you to start and stop the Sun GlassFish Web Space Server (GlassFish server) from within the IDE using the generic server framework in Eclipse.
- OpenPortal Portlet Container — This plug-in registers the OpenPortal Portlet Container 2.0 that runs on GlassFish server in Eclipse IDE, as a Server Runtime. Currently, the OpenPortal Portlet Container plug-in allows you to start and stop the Portlet Container (GlassFish server) from within the IDE using the generic server framework in Eclipse.

Installing the Eclipse Portal Pack

You need to install the Eclipse Portal Pack to use the plug-ins and develop and deploy portlets.

▼ To Install the Eclipse Portal Pack

1. **Download Eclipse Portal Pack from the Eclipse Portal Pack download page.**
2. **Unzip the Eclipse Portal Pack.**
3. **Copy the `.jar` files to the *ECLIPSE_HOME*`\plugins` directory.**
4. **(Optional) — Restart Eclipse, only if it is running while you copy the `.jar` files.**

Using the Eclipse Portal Pack

This section explains how to use the Eclipse Portal Pack and the plug-ins in the Sun GlassFish Web Space Server. You need to perform the following tasks before using the Eclipse Portal Pack.

1. Creating Runtime and Server for the Sun GlassFish Web Space Server.
2. Creating a Runtime and Server for OpenPortal Portlet Container 2.0

Creating a Runtime and Server for the Sun GlassFish Web Space Server

You need to create a runtime and server for Eclipse Portal Pack in order to deploy it on the Sun GlassFish Web Space Server.

▼ To Create a Runtime and Server for the Sun GlassFish Web Space Server

1 On Eclipse IDE, click the Window menu and select Preferences.

The Preferences window appears.

2 From the left pane, click Installed Runtimes.

The Installed Server Runtime Environments appear on the right pane.

3 Click Add.

The New Server Runtime window appears. In this window, you can choose the runtime environment for a newly installed server.

4 From the available runtime types, expand Sun Microsystems Inc and select Sun GlassFish Web Space Server. Select the Also create new local server check box.

5 Click Next.

6 Enter the JRE name, GlassFish Home directory, and the Domain Name information for Sun GlassFish Web Space Server.

These details are useful to create a runtime for Sun GlassFish Web Space Server.

7 Click Next.

8 Enter the server address, server port, and the debug port information for Sun GlassFish Web Space Server.

9 Click Finish.

The Sun GlassFish Web Space Server server is created.

Creating a Runtime and Server for OpenPortal Portlet Container 2.0

You need to create a runtime and server for Eclipse Portal Pack in order to deploy it on OpenPortal Portlet Container 2.0.

▼ To Create a Runtime and Server for OpenPortal Portlet Container 2.0

1. **On Eclipse IDE, click the Window menu and select Preferences.**

   The Preferences window appears.

2. **From the left pane, click Installed Runtimes.**

   The Installed Server Runtime Environments appear on the right pane.

3. **Click Add.**

   The New Server Runtime window appears. In this window, you can choose the runtime environment for a newly installed server.

4. **From the available runtime types, expand Sun Microsystems Inc and select OpenPortal Portlet Container 2.0. Select the Also create new local server check box.**

5. **Click Next.**

6. **Enter the JRE name, GlassFish Home directory, and the Domain Name information for OpenPortal Portlet Container 2.0.**

   These details are useful to create a runtime for OpenPortal Portlet Container 2.0.

7. **Click Next.**

8. **Enter the server address, server port, and the debug port information for OpenPortal Portlet Container 2.0.**

9. **Click Finish.**

   The OpenPortal Portlet Container 2.0 server is created.

Creating and Deploying a Portlet

You can use the plug-ins in Eclipse Portal Pack to create and deploy portlets on Sun GlassFish Web Space Server. This section explains how to create and deploy portlets on Sun GlassFish Web Space Server using the Eclipse Portal Pack plug-ins.

▼ To Create and Deploy a Portlet

1 **On Eclipse IDE, click the File menu and select New. Form New, choose Dynamic Web Project.**

The New Dynamic Web Project window appears.

2 **In the New Dynamic Web Project window, enter a name for your project, choose Sun GlassFish Web Space Server or OpenPortal Portlet Container 2.0 as project runtime, and select Portlet 1.0 or 2.0 support as the configuration.**

Note – If Portlet 2.0 configuration is not available in the list, you need to manually add the configuration. To do this, click Modify, select Portal Pack and click Save.

---

3 **Click Finish.**

A new web project is created.

4 **To add a portlet to the web project, click the File menu. From the options, select New and click Other.**

The New window to create a Java portlet appears.

5 **From the list of Wizards, expand the Portlet wizard and click Java Portlet.**

6 **Click Next.**

The Create a Portlet Class window appears.

7 **Enter the details of the portlet that you want to generate.**

8 **Click Finish.**

A portlet class is created and an entry about the portlet is created in the `portlet.xml` file.

9 **To deploy the portlet, right click on the name of the portlet, select Run As, and click Run On Server. Accept the default values.**

This action deploys the portlet web application and starts the target server, which is either Sun GlassFish Web Space Server or OpenPortal Portlet Container.

Developing Workflows Using Simple API for Workflow in Sun GlassFish Web Space Server

The Sun GlassFish Web Space Server is bundled with the Simple API for WorkFlow (SAW) feature. This section explains about the Simple API for WorkFlow (SAW) feature and how you can develop workflow portlets using NetBeans 6.5 IDE or higher versions. The topics covered in this section are:

- Introduction to SAW
- Prerequisites
- Adding SAW plug-in in NetBeans IDE
- Creating a Web Application Using the Drag and Drop Features in SAW plug-in
- Creating a SAW Portlet Using the SAW plug-in
- Deploying the Workflow Portlet an Open Source Portlet Container
- Deploying the Workflow Portlet on Sun GlassFish Web Space Server
- Using the Workflow Portlet
- References

Introduction to SAW

SAW is a generic workflow API to perform human workflow interaction with various workflow engines. SAW provides a framework to plug-in any business process specific implementation that implements SAW interfaces. SAW is shipped with a default implementation for Sun Java™ Composite Application Platform Suite. You can have similar SAW implementations for other business processes such as JBoss Business Process Management (jBPM) and Open Enterprise Service Bus (OpenESB).

Prerequisites

Before you build an application using SAW APIs, you must install and configure the following components:

- Java Development Kit (JDK™) 1.5 or later versions from `http://java.sun.com/javase/downloads/index.jsp`.
- Sun Java Composite Application Platform Suite. Currently, SAW implementation is available only for Sun Java Composite Application Platform Suite. Therefore, a Sun Java Composite Application Platform Suite installation is required. If you need any other OpenSource workflow engine implementation, visit `https://saw.dev.java.net/`, for regular updates.
- NetBeans 6.5 from `http://download.netbeans.org/netbeans/6.0/final/`.
- Portal Pack 3.0 from `http://portalpack.netbeans.org/`

- Sun GlassFish Web Space Server from `https://portlet-container.dev.java.net/`
- SAW plug-in for NetBeans from `http://portalpack.netbeans.org/`

Before proceeding with the implementation of SAW in your application, you need to configure a business process using Sun Java Composite Application Platform Suite. For more information how to create a business process in Sun Java Composite Application Platform Suite, read the article `http://developers.sun.com/portalserver/reference/techart/workflow.html`

Creating a Web Application Using the Drag and Drop Features in SAW plug-in

This section provides a procedure to explain how to create a simple web application in NetBeans using the drag and drop features available in SAW plug-in.

▼ To Create a Web Application Using the Drag and Drop Features in SAW plug-in

1. **Click New Project in NetBeans IDE.**
2. **Select Web from Categories and Web application from Projects and click Next.**
3. **Enter a project name.**
4. **From the Server drop down list, select GlassFish v2 and click Next.**
5. **From the Frameworks list box, select the SAW plug-in.**

   You can see that the Java Composite Application Platform Suite is selected by default, for the SAW implementation
6. **Click Finish.**

Tip – Check the SAW project web site `https://saw.dev.java.net/` regularly, for updates on these implementations.

---

7. **Expand the Source Packages folder to see the `ImplementationType.properties` and `WorkflowConfig.properties` property files, under the default package folder.**

   The `ImplementationType.properties` file sets the ImplementationType as Java Composite Application Platform Suite. This can be changed in future saw releases to point to JBPM, OS Workflow or other workflow Engines.

8 **Set the `WorkflowConfig.properties` file appropriately, to point the Java Composite Application Platform Suite Integration server.**

The web application is listed in the Projects tab and the `index.jsp` appears in the code window. You can see the appropriate Workflow tags listed in the Palette.

Note – If the Workflow tags palette is not loaded in the NetBeans IDE, you can manually load it by clicking Tools and selecting Workflow from the Palette option. Moreover, if you right click on the web application and select Properties, the Libraries Category in the Project Properties window displays the SAW libraries that are added by the SAW plug-in in the NetBeans IDE. The current version of SAW libraries is 0.8. Check the SAW project web site `https://saw.dev.java.net/` regularly for updates on the plug-in for new versions of SAW.

---

9 **Open the Java file in your web application.**

You can notice that the relevant Workflow APIs are listed in the palette. You can drag and drop these Workflow APIs in the Java file, depending upon the usage for your logic. In the default package, you can see various properties required by SAW to execute. These properties are the location of Sun Java Composite Application Platform Suite, SAW implementation and so on.

Note – You cannot drag and drop duplicate methods. There are two ways to use the SAW APIs. One way is to use through JSP tag libraries and the other way is through the Java APIs.

---

10 **Edit the `WorkflowConfig.Properties` file to define your runtime execution environment.**

The web application is ready to deploy. The sample `WorkflowConfig.Properties` file is shown below:

```
# the business process to use.
sawworkflowimplclass = The SAW Workflow Implementation class to use. For example,
com.sun.saw.impls.jcaps.JCAPSWorkflow
# Properties that are needed by the JCAPS Implementation of SAW.
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverhost = machine name or IP where
Sun Java Composite Application Platform Suite business process is running.
For example, abc.india.sun.com
com.sun.saw.impls.jcaps.JCAPSWorkflow.iiopserverport =
Port on which the IIOP lookup for EJB happens. For example, 18002
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverusername =
Administrator user name. For example, Administrator
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverpassword =
Administrator password of JCAPS Integration server. For example, <password>
com.sun.saw.impls.jcaps.JCAPSWorkflow.contextfactory =
The context factory. For example, com.sun.jndi.cosnaming.CNCtxFactory
com.sun.saw.impls.jcaps.JCAPSWorkflow.serviceJndi =
The JNDI look up name. For example, WorkflowService
```

Caution – Do not edit the `ImplementationType.Properties` file.

11. **Right click on the created web application and select Libraries.**
12. **Click Add Jar and select the `WorkflowServiceClient.jar` from Java Composite Application Platform Suite and click OK.**
13. **Right click on the web application and select Clean and Build.**
14. **Right click on the web application and select Run.**

    A web browser is launched and it displays the user interface defined in your web application.

Creating a SAW Portlet Using the SAW plug-in

This section explains you how to create a simple workflow portlet in NetBeans IDE using the SAW plug-in.

▼ To Create a SAW Portlet Using the SAW plug-in

1. **Click New Project in NetBeans IDE.**
2. **Select Web from Categories and Web application from Projects and click Next.**
3. **Enter a project name.**
4. **From the Server drop down list, select Open Portal Portlet Container 2.0 Beta and click Next.**
5. **From the Frameworks list box, select Portlet Support and select the Create Portlet checkbox.**
6. **Select the SAW plug-in and click Finish.**
7. **Expand the Source Packages folder to see the `ImplementationType.properties` and `WorkflowConfig.properties` property files, under the default package folder.**

   The `ImplementationType.properties` file sets the ImplementationType as Java Composite Application Platform Suite. This can be changed in future saw releases to point to JBPM, OpenESB or other workflow Engines
8. **Set the `WorkflowConfig.properties` file appropriately, to point the Java Composite Application Platform Suite Integration server.**

9 **Edit the `WorkflowConfig.Properties` file to define your runtime execution environment.**

The sample `WorkflowConfig.Properties` file is shown below:

```
# the business process to use.
sawworkflowimplclass = The SAW Workflow Implementation class to use.
For example, com.sun.saw.impls.jcaps.JCAPSWorkflow
# Properties that are needed by the JCAPS Implementation of SAW.
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverhost = machine name or IP
where Sun Java Composite Application Platform Suite business process is running.
For example, abc.india.sun.com
com.sun.saw.impls.jcaps.JCAPSWorkflow.iiopserverport =
Port on which the IIOP lookup for EJB happens. For example, 18002
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverusername =
Administrator user name. For example, Administrator
com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverpassword =
Administrator password of JCAPS Integration server. For example, <password>
com.sun.saw.impls.jcaps.JCAPSWorkflow.contextfactory = The context factory.
For example, com.sun.jndi.cosnaming.CNCtxFactory
com.sun.saw.impls.jcaps.JCAPSWorkflow.serviceJndi = The JNDI look up name.
For example, WorkflowService
```

10 **Change the portlet Java and the JSP files in your web application accordingly, to write the portlet.**

You can notice that the relevant Workflow APIs are listed in the palette that you can drag and drop in the Java file, depending upon the usage for your logic.

11 **Right click on the created portlet application and select Libraries.**

12 **Click Add Jar and select the `WorkflowServiceClient.jar` from Java Composite Application Platform Suite and click OK.**

13 **Right click on the web application and select Clean and Build.**

14 **Right click on the web application and select Run.**

The portlet gets rendered on the portlet container.

Deploying the Workflow Portlet an Open Source Portlet Container

You need to deploy the Workflow portlet in order to work with the Workflow API. This section provides a procedure to explain how to deploy the workflow portlet on Open Source Portlet Container.

▼ To Deploy the Workflow Portlet on Open Source Portlet Container

1. **Download the workflow portlet from the SAW project web site** `https://saw.dev.java.net/`**.**

2. **Extract the** `workflowPortlet.war` **into a local directory using the** `jar -xvf workflowPortlet.war` **command.**

3. **Modify the** `WorkflowConfig.properties` **file in the** `\workflowPortlet\WEB-INF\classes` **directory by providing suitable values to the following properties:**

   - `sawworkflowimplclass = com.sun.saw.impls.jcaps.JCAPSWorkflow`. This means that you are using SAW Workflow implementation class, `JCAPSWorkflow`.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverhost =` *machine name* or *IP address*, where Sun Java Composite Application Platform Suite business process is running. For example, *abc.india.sun.com*.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.iiopserverport =` *port* on which the IIOP lookup for EJB happens. For example, *18002*.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverusername =` *Administrator user name*. For example, *Administrator*.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverpassword =` *Administrator password*. For example, *xxx*.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.contextfactory =` *The context factory*. For example, *com.sun.jndi.cosnaming.CNCtxFactory*.
   - `com.sun.saw.impls.jcaps.JCAPSWorkflow.serviceJndi =` *The JNDI look up name*. For example, *WorkflowService*.

4. **Modify the** `workflowportlet.properties` **file in** `\workflowPortlet\WEB-INF\classes` **directory by providing appropriate values to the properties. The sample** `workflowportlet.properties` **file to deploy on Portlet Container is shown below.**

   authenticationRepository = accessManager. Specify the authentication repository that is used.

   Note – accessManager is the default value that is provided for authenticationRepository in the `workflowportlet.properties` file. Only if you want to deploy the `.war` file using the Open Source Portlet Container, then change the authenticationRepository value to appServer.

   ---

5. **Ensure that you have the client stubs (**`WorkflowServiceClient.jar`**) generated out of the workflow service, deployed on Sun Java Composite Application Platform Suite. Copy this to** `\workflowPortlet\WEB-INF\lib` **directory by using the** `cp WorkflowServiceClient.jar workflowServiceClient-1.0.jar` **command.**

6. **Recreate the** `.war` **file using the** `jar –cvf` **command.**

7 **Deploy the Workflow portlet using the admin tab of the Portlet Server Administration Console.**

To deploy the workflow portlet on Open Source Portlet Container:

- Create a user under admin-realm of GlassFish V2 or Application Server 9.1, on which the Open Source Portlet Container is running. For example, create a user, CPina in the admin-realm of GlassFish V2.
- Login to the GlassFish V2 admin.
- Navigate to Configuration -> Security -> Realms.
- Click admin-realm and Click Manage Users.
- Add CPina, group list : asadmin and passowrd as CPina.
- Access the portlet by typing `http://`*machine*`:`*port*`/portal/dt` in a web browser.
- Log in as user, CPina. The tasks that are assigned to CPina will be displayed on the workflow portlet.

If it is a web application, then the `saw-api-0.6.jar` should be in the appropriate classpath. The `saw.tld`, `saw-impl-jcaps-0.6.jar`, and `WorkflowServiceClient.jar` should be bundled with the application.

8 **Modify the `jreHome\lib\logging.properties` file. To do this, update the `java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter` property. The logging levels can be configured here. For example, INFO, ERROR and so on.**

Note – You need not modify the `logging.properties` file in the case of OpenPortal. For stand alone Java applications, you need to modify the `logging.properties` file to specify the type of format and the level of log.

Deploying the Workflow Portlet on Sun GlassFish Web Space Server

You need to deploy the Workflow portlet in order to work with the Workflow API. This section provides a procedure to explain how to deploy the workflow portlet on Open Portal 7.2.

▼ To Deploy the Workflow Portlet on Sun GlassFish Web Space Server

1 **Extract the `workflowPortlet.war` into a local directory using the `jar -xvf workflowPortlet.war` command.**

Note – The workflow portlet is available in the Sun GlassFish Web Space Server install under the `/opt/sun/portal/portlet/` directory for Linux and `/opt/SUNWportal/portlet/` directory for Solaris. You can also download the workflow portlet from the Portlet Repository `http://wiki.java.net/bin/view/OpenPortal/PortletsInTheRepository`.

---

2 **Ensure that you have the client stubs (`WorkflowServiceClient.jar`) generated out of the workflow service, deployed on Sun Java Composite Application Platform Suite. Copy this to `\workflowPortlet\WEB-INF\lib` directory by using the `cp WorkflowServiceClient.jar workflowServiceClient-1.0.jar` command.**

3 **Recreate the `.war` file using the `jar –cvf` command.**

4 **Deploy the Workflow portlet using the admin tab of the Portal Server Administration Console.**

To deploy the workflow portlet on Open Portal:

- Create a channel with this portlet provider and add it to one of the tab
- Click the created new channel created and change the following portlet preferences:
  - `sawworkflowimplclass` = com.sun.saw.impls.jcaps.JCAPSWorkflow. This means that you are using SAW Workflow implementation class, JCAPSWorkflow.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverhost` = machine name or IP address, where Sun Java Composite Application Platform Suite business process is running. For example, `abc.india.sun.com`.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.iiopserverport` = port on which the IIOP lookup for EJB happens. For example, 18002.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverusername` = Administrator user name. For example, Administrator.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.appserverpassword` = Administrator password. For example, xxx.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.contextfactory` = The context factory. For example, com.sun.jndi.cosnaming.CNCtxFactory.
  - `com.sun.saw.impls.jcaps.JCAPSWorkflow.serviceJndi` = The JNDI look up name. For example, WorkflowService.

Note – accessManager is the default value that is provided for authenticationRepository in the `workflowportlet.properties` file. You need not change this property, if the portlet is deployed on Open Portal.

---

5 **Create a user with userId as CPina in Access Manager with the status as active. Assume that the Business process assigns the newly created tasks to this user.**

6 **Access the portlet by typing `http://machine:port/portal/dt` in a web browser.**

7 **Log in as user CPina. The tasks that are assigned to CPina will be displayed on the workflow portlet.**

Only in the case of Open Portal, the `saw-api-0.6.jar` should be available in the server classpath. If it is a web application, then the `saw-api-0.6.jar` should be in the appropriate classpath. The `saw.tld`, `saw-impl-jcaps-0.6.jar`, and `WorkflowServiceClient.jar` should be bundled with the application.

Using the Workflow Portlet

1. Login to the Portal Server 7.2 desktop (`http://portalmachine:port/portal`) with username and password as CPina.
2. As soon as the user logs in, all the tasks assigned to the user are displayed. The page contains three sections.
   - First section — Search Criteria section, where the user can mention the criteria based on which the tasks have to be filtered and displayed.
   - Second section — Task List section that displays all the tasks that match the search criteria of a user.
   - Third section — Displays the buttons, which are the operations that can performed on any task.
3. The search criteria section has the following attributes using which the user can filter tasks:
   - Start Date and End date - Filter tasks based on a valid start date and end date.
   - TaskIds - If the user knows the taskids, then the user can enter them in the text area that is separated by commas.
   - UserIds - Filter tasks based on the current owner of the task.
   - Task Status - Filter tasks based on the status of the task. For example, pending or completed.
   - GroupId - Filter tasks based on the group id. For example, Vice President Marketing and so on.
   - Flex String Type - Filter tasks based on a flex string type. For example, flexString1, flexInt1, flexDouble1 and so on.
   - Flex String Value - The value of the flex string type that is selected by the user.
4. When the user clicks the Search button, tasks are filtered based on the search criteria that the user has entered and displayed in the "Task List" section.
5. The "Clear" button clears any of the search criteria that the user has entered or selected.
6. Assume that a new task is allocated to the user "CPina". Then “CPina” can do the following operations on the task that is assigned:

- *EXECUTE* - Select the task and click the Execute button. A new page for executing a task is displayed. This page shows the input for the task in a text box and flex string in another text box. The user can update both the text boxes and click the Save button. These values get updated for the task. On executing, the task gets checked out on the name of the logged in user (CPina), who is the current owner of this task. Currently, the task can be executed only once. This is because, on execution, the task is checked out, and a task can be checked out only once by a given user. If the user tries to execute the second time, then "Task Exception" error message will be displayed.
- *COMPLETE* - Select the task and click the Complete button. Now, the status of the task is marked as completed. The user has to be the current owner for performing this operation on this task.
- *CHECKIN* - Select the task and click the `CheckIn` button. This operation is opposite of check out. The currently logged in user is not the current owner now. The user has to be the current owner for performing this operation on this task.
- *HISTORY* - Select the task and click the `History` button. On clicking this, the task history is shown in a new JSP. All operations that were performed on the task is displayed. Click the `Back` button on the task history JSP to come back to the task list page.
- *ESCALATE* - Select the task and click the `Escalate` button. On clicking this, the task gets assigned to the manager of the currently logged in user. The user has to be the current owner for performing this operation on this task.
- *REASSIGN* - Select users from the `Reassign selected task to` drop down, and click the `Reassign` button, the task gets assigned to these users. The user has to be the current owner for performing this operation on this task.
- *DELETE* - Select the task and click the `Delete` button. On this operation, the task is deleted. The user has to be the current owner for performing this operation on this task.

7. Pagination support is provided for the Task List. On the first page the `Previous` link is not displayed. On the last page, the `Next` link is not displayed. On all the other pages, both the `Previous` and `Next` link is displayed.
8. Any Error messages are displayed in the top of the portlet in red font.
9. Any confirmation messages are displayed in the top of the page in blue font.
10. The attributes that can be modified using the portlet preferences are the `noOfrecords` and the `executeJSP`. You need to click the edit mode to change the values for the portlet preferences.
11. All the messages that are displayed are internationalized.

References

For more information on SAW implementation, refer the following resources:

- For regular updates on SAW project, visit the SAW project web site `https://saw.dev.java.net/`.
- For more information on how to incorporate workflow in Sun Portal Server with Sun Java Composite Application Platform Suite, read the article `http://developers.sun.com/portalserver/reference/techart/workflow.html`.
- For more information on how to work with SAW in NetBeans 6.0 IDE, check this screencast.
- Download JavaDocs for SAW API from here `https://saw.dev.java.net/docs/SAWDownloadPage.html`.
- For regular updates on Portal Pack project, visit the Portal Pack project web site `http://portalpack.netbeans.org/`.
- For regular updates on OpenPortal Portlet Container project, visit the Portlet Container project web site `https://portlet-container.dev.java.net/`.
- For more information on NetBeans and the most recent downloads available, visit the NetBeans web site `http://www.netbeans.org/`.

Developing Themes Using the ViewDesigner Plug-in in Sun GlassFish Web Space Server

The ViewDesigner plug-in available in Sun GlassFish Web Space Server, enables web designers to design new and customize existing themes, which could then be applied to a portal page.. You can download the ViewDesigner plug-in from the ViewDesigner project web site. The topics covered in this section are:

Overview of ViewDesigner Plug-in

The layout and design of web or portal page(s) is done by professional web designers. Web designers are comfortable with tools such as Dreamweaver, Microsoft FrontPage and so on. Most portals at present, provide a web based user interface to create, design, and theme a portal. So, it would be better and an advantage, if portals provide a method by which a web designer is able to use professional web design tools to create an elegantly designed portal.

The ViewDesigner plug-in makes it possible for a web designer to create a well designed user interface for a portal in Sun GlassFish Web Space Server. The ViewDesigner project aims to provide plug-ins for popular web design tools, which enables a web designer to easily customize, design, and theme a portal page. For more information on ViewDesigner project, see the project home page.

At present, the Sun GlassFish Web Space Server provides a plug-in for DreamWeaver 8, Adobe Dreanweaver CS3, and Adobe Dreamweaver CS4.

Note – Currently, the ViewDesigner plug-in is supported only on Windows platform.

Installing the Dreamweaver Software

Before installing the ViewDesigner plug-in, you need to install Dreamweaver software on your desktop.

▼ To Install Dreamweaver Software

1. **Download Dreamweaver software from Adobe downloads page.**
2. **Follow the installation instructions and install the software.**
3. **While installing the Dreamweaver software, ensure that you also install Dreamweaver Extension Manager.**

Installing the ViewDesigner Plug-in

You need to install the ViewDesigner plug-in on the Dreamweaver software to create new or modify existing themes in the Sun GlassFish Web Space Server.

▼ To Install the ViewDesigner Plug-in

1. **Download the ViewDesigner plug-in. The most recent version of the plug-in is available on the ViewDesigner downloads page.**
2. **Install the ViewDesigner plug-in using the Dreamweaver Extension Manager. To do this:**
   a. **Form the Dreamweaver Extension Manager, click File and select Install Extension.**
   b. **Click Browse and select the .mxp file that you have downloaded.**
   c. **Click OK.**
   d. **Accept the license agreement.**
      A message appears that the ViewDesigner plug-in has been installed successfully.
3. **Open the Dreamweaver software.**
4. **To confirm the successful installation, click File and select the Import and Export options.**
   You should see the Import Web Space Theme File and Export Web Space Theme File options.

Downloading an Existing Theme WAR File

In order to modify an existing theme, you need to download a theme.

▼ To Download an Existing Theme WAR FIle

1. **Using the Sun GlassFish Web Space Server Plug-in Installer portlet, download any existing Theme WAR file. To do this:**
   a. **On Sun GlassFish Web Space Server, add the Plug-in Installer portlet.**
   b. **From the Plug-in Installer portlet, click the Browse Repository tab.**
   c. **Click the Themes Plugins tab and download an existing theme.**
2. **Alternatively, you could download a Theme WAR file from the official or community plug-ins page on Liferay or the SourceForge repository.**

Importing the Theme WAR File into Dreamweaver Software

After downloading the Theme WAR file, you need to import the file into the Dreaweaver software to modify the theme.

▼ To Import the Theme WAR File into Dreamweaver Software

1. **From the Dreamweaver software, click File, select Import, and select Import Web Space Theme File option.**

   The Import Web Space Theme File window appears.

2. **Enter appropriate values in the following fields.**

   - Path to Web Space Theme file — Specify the location of the Sun GlassFish Web Space Server Theme WAR file that you have downloaded using the Plug-in Installer portlet.
   - Working folder of Web Space Theme file — Specify any directory on your desktop where you want to extract the Theme WAR file.
   - Path to `Java.exe` — Specify the location of `Java.exe` file. Once you mention this location, this value gets persisted, and shows up in this text box by default. If you do not want this value, you can change it.

3. **(Optional) You may click Cancel or Help to close the Import Web Space Theme File window or to open the Help for importing the Theme WAR file.**

4. **Click OK.**

   The Theme WAR file is imported into the Dreamweaver software.

   The ViewDesigner plug-in extracts the Theme WAR file in the specified working folder and opens a sample preview page (`index.html`). This is just a preview page, and the changes made to this page does not appear on the portal.

Modifying the Theme File

Modify the changes in the Theme file according to your requirements. The `index.html` page shows you a preview of the changes that you are making to the CSS files. After doing the necessary modifications, click File and select Save or Save All to save the changes to the Theme file.

Exporting the Modified Theme WAR File

After doing the modifications in the Theme WAR file, you need to export the file to a new location.

▼ To Export the Modified Theme WAR File

1 **From the Dreamweaver software, click File, select Export, and select Export WebSynergy Theme File option.**

The Export WebSynergy Theme File window appears.

2 **Enter appropriate values in the following fields.**

- Working Folder for WebSynergy Theme file — Specify the location of the folder where you have extracted the Theme WAR file and made changes.
- Save New WebSynergy Theme file To — Specify the location where you would like to save the modified Theme WAR file.
- Path to `Java.exe` — Specify the location of `Java.exe` file.

Note – As soon as you specify the working folder for WebSynergy Theme file, the ViewDesigner plug-in reads the `liferay-look-and-feel.xml` and `liferay-plugin-package.properties` files and populates the fields in the Theme Properties:. You might want to change the filed values or leave them as it is before the Export operation. If you want to change any of the theme properties, you can enter the new values, and it gets persisted into the respective files.

- Theme Id — The id of the theme. This gets persisted into the `liferay-look-and-feel.xml` file.
- Theme Name — The name of the theme. This is the name that gets displayed in the list of available themes. This gets persisted into the `liferay-look-and-feel.xml` and `liferay-plugin-package.properties` files.
- Theme Description — A short description of the theme. This gets persisted into the `liferay-plugin-package.properties` file.
- Compatibility Version — The version of Sun GlassFish Web Space Server 10.0, this theme is compatible with. This gets persisted into the `liferay-look-and-feel.xml` file.
- Author — The author (for example, company name) of the theme. This gets persisted into the `liferay-plugin-package.properties` file.
- Author URL — The author URL (could be the company URL). This gets persisted into the `liferay-plugin-package.properties` file.
- Theme Licence — The licence of the theme. This gets persisted into the `liferay-plugin-package.properties` file.

3 **(Optional) You may click Cancel or Help to close the Export WebSynergy Theme File window or to open the Help for exporting the Theme WAR file.**

4 **Click OK.**

The Theme WAR file is exported to the Sun GlassFish Web Space Server 10.0 Theme WAR. A message appears to inform you that the export was successful.

Deploying the New Theme WAR File

After exporting the modified Theme WAR file, you need to deploy the file on Sun GlassFish Web Space Server.

▼ To Deploy the New Theme WAR File

1 **In the Plugin Installer portlet, click the Upload File tab.**

2 **Click Browse for the new WAR file and select the modified Theme WAR file.**

3 **Click Install.**

The new Theme WAR file is deployed on Sun GlassFish Web Space Server.

4 **Alternatively, you can upload the new Theme WAR file into the Sun GlassFish Web Space Server Hot Deploy directory to automatically deploy the new Theme WAR file on Sun GlassFish Web Space Server.**

Applying the New Theme to a Sun GlassFish Web Space Server Community

You can apply the newly created Theme to any community on Sun GlassFish Web Space Server.

▼ To Apply the New Theme to Sun GlassFish Web Space Server Community

1 **On Sun GlassFish Web Space Server, select a community.**

2 **From the Welcome drop down menu, click Manage Pages and select Look and Feel.**

The newly deployed Theme appears.

3 **Click the new Theme and set it as the current theme for the selected community.**