An Oracle White Paper
September 2009

Exposing WebCenter Services Task Flows as WSRP Portlets and Ensemble Pagelets

Disclaimer

Table of Contents

Introduction

This document serves as a guide for application developers wanting to expose WebCenter service task flows as WSRP portlets or Ensemble pagelets, and reflects recommended best practices. To help understand the process, we use the WebCenter Document Library service task flow as an example throughout this document. The following outlines the high level tasks:

- Create a custom WebCenter application and test the service task flows
- Create producer endpoints:
    - WSRP portlet endpoint
        - Portletize pages containing WebCenter services task flows
        - Add support for portlet geometry parameters
        - Wire WebCenter services task flow parameters to portlet parameters
        - Add Trinidad pop-up support
    - Ensemble pagelet endpoint
        - Add the Ensemble endpoint servlet filter
        - Add support for pagelet geometry parameters
        - Wire WebCenter services task flow parameters to URL parameters
- Make the producer session cookie unique

The resulting WSRP portlets can be consumed by custom Oracle WebCenter applications, by Oracle Portal 11g, and by Oracle WebLogic Portal. The Ensemble pagelets can be consumed by Oracle WebCenter Ensemble and Oracle WebCenter Interaction.

In addition to this guide, a sample JDeveloper workspace is provided. To use the workspace, unzip the SampleServicesProducer.zip and open the ServicesProducers/ServicesProducer.jws in JDeveloper. This guide makes references to various classes within the sample workspace.

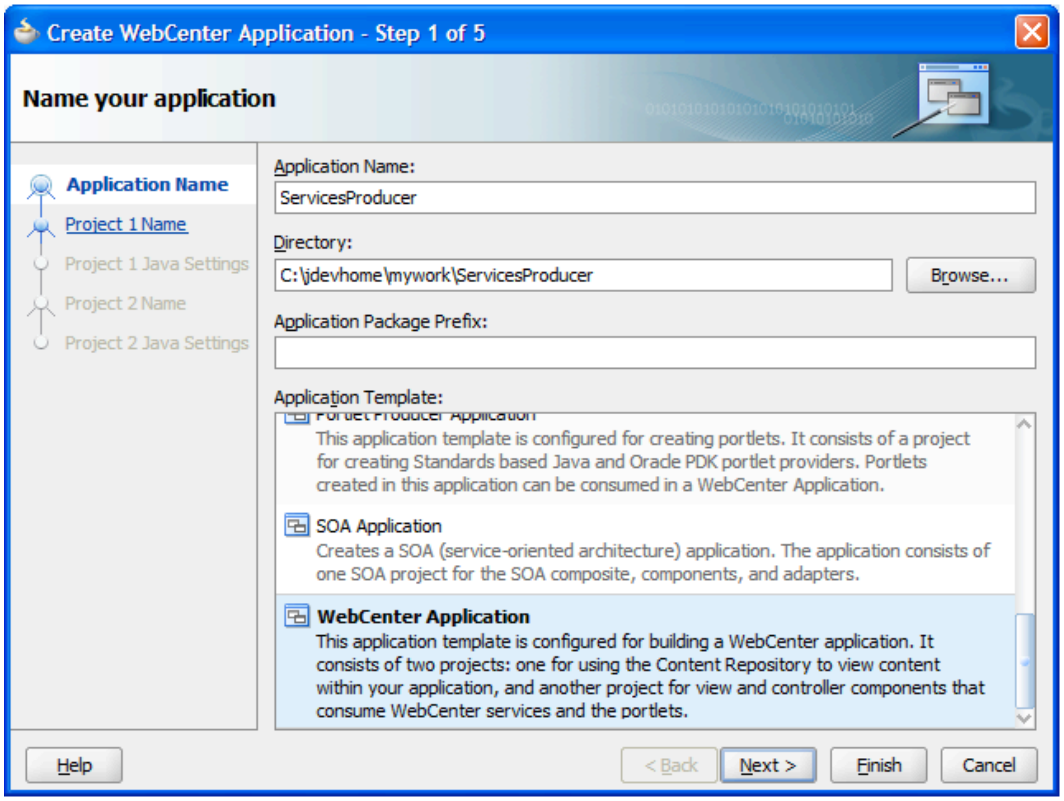
Creating a WebCenter Application and Testing the Service Task Flows

Before turning a WebCenter service task flow into a portlet or pagelet, we recommend using the service task flow on an ADF Faces page to check that the task flow functions as intended. Depending on the intended use, one or multiple WebCenter service task flows can be combined in a single portlet or pagelet.

Create a WebCenter Application

Follow the instructions provided in “Creating a WebCenter Application” in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*, to create an application in JDeveloper based on the WebCenter Application Template. Figure 1 illustrates the Create WebCenter Application wizard used for this task.

Figure 1 - Create WebCenter Application Wizard



Application Name: ServicesProducer

Directory: C:\jdevhome\mywork\ServicesProducer

Application Package Prefix:

Application Template:

Portlet Producer Application
This application template is configured for creating portlets. It consists of a project for creating Standards based Java and Oracle PDK portlet providers. Portlets created in this application can be consumed in a WebCenter Application.

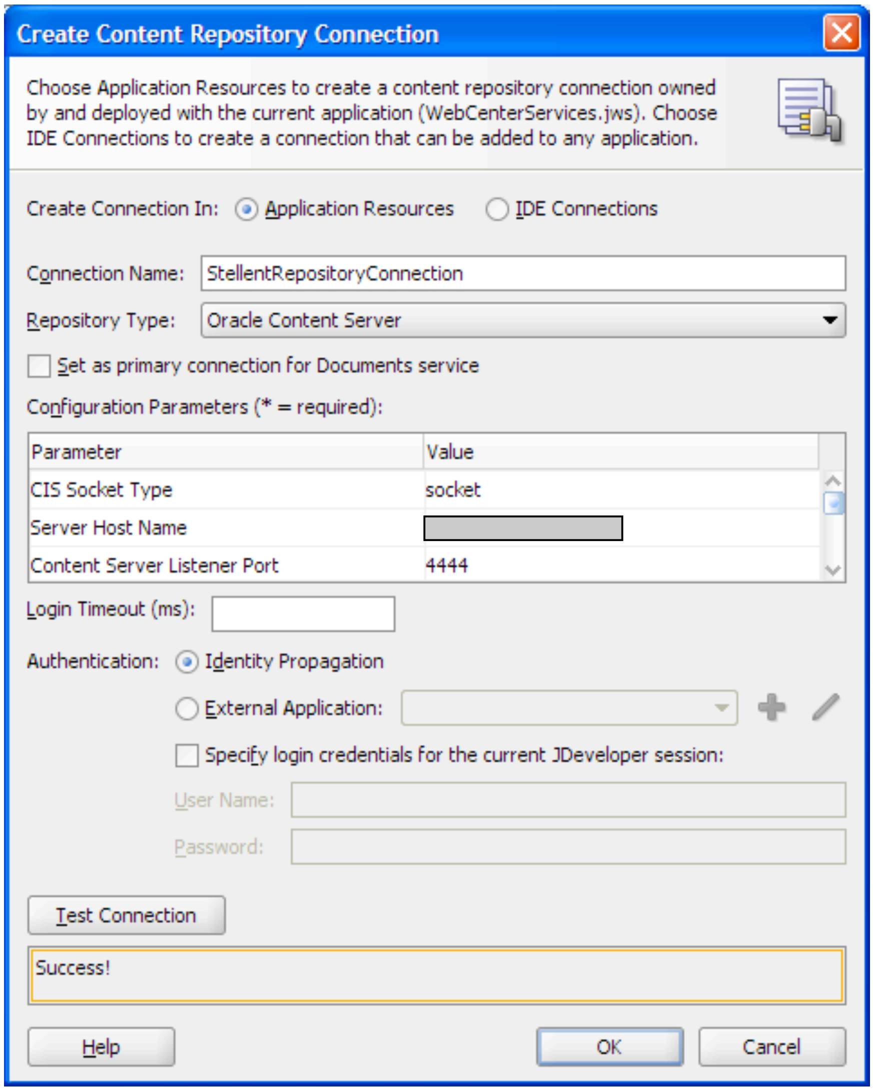
SOA Application
Creates a SOA (service-oriented architecture) application. The application consists of one SOA project for the SOA composite, components, and adapters.

WebCenter Application
This application template is configured for building a WebCenter application. It consists of two projects: one for using the Content Repository to view content within your application, and another project for view and controller components that consume WebCenter services and the portlets.

Create Backend Server Connections

Follow the instructions provided in “Accessing Connection Wizards” of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*, to create the required backend server connections for the services. Figure 2 shows example settings for the Create Content Repository Connection wizard.

Figure 2 - Create Content Repository Connection Wizard



Create Content Repository Connection

Choose Application Resources to create a content repository connection owned by and deployed with the current application (WebCenterServices.jws). Choose IDE Connections to create a connection that can be added to any application.

Create Connection In: (●) Application Resources ( ) IDE Connections

Connection Name: StellentRepositoryConnection

Repository Type: Oracle Content Server

[ ] Set as primary connection for Documents service

Configuration Parameters (* = required):

Parameter	Value
CIS Socket Type	socket
Server Host Name	[redacted]
Content Server Listener Port	4444

Login Timeout (ms):

Authentication: (●) Identity Propagation

( ) External Application:

[ ] Specify login credentials for the current JDeveloper session:

User Name:

Password:

Test Connection

Success!

Help OK Cancel

For development and testing, we recommend using the **External Application** option with public or shared credentials to access the backend servers. For deployment to a production environment, we recommend using the **Identity Propagation** option. Note that the backend connections for a development environment are usually different from those used in a production environment. The connections created in JDeveloper can be safely changed prior to deployment in JDeveloper, alternatively during or after application deployment using the Fusion Middleware Control.

Add Service Task Flows to the Page and Test

Create a JSF page and drop an <af:panelStretchLayout> onto the page inside the <af:form> tag. The panelStretchLayout serves as a geometry container for the service task flow. Next, drop the desired service task flow into the panelStretchLayout's center facet. Since this page is used to render the content of the portlet and pagelet, it is essential to run the page to make sure the task flow works as intended. An example JSF page, *doclib-document-library.jspx*, is provided in the sample workspace.

Turning the Application into a Portlet Producer

Create the Portlets

By this point, the service task flow should be running as intended on a page. The page is now ready to be turned into a portlet. Follow the instructions in the Creating a Portlet from a JSF Application chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter* to portletize the page you have just created. The *How to Create a JSF Portlet Based on a Page* section provides step-by-step instructions for creating a portlet.

Add support for portlet geometry parameters

It's generally useful to control the dimension of a portlet on the page using the custom portlet parameters *width* and *height*. This section describes how to add these parameters, and use them to control the portlet dimensions.

First, declare the width and height parameters with the portlet. To do this, add the parameters to the `WEB-INF/oracle-portlet.xml` file of the producer. For example:

```
<portlet-extension>
    <portlet-name>DocumentLibraryPortlet</portlet-name>
    <navigation-parameters>
        <name>width</name>
        <type>String</type>
        <label xml:lang="en">width</label>
        <hint xml:lang="en">Portlet Width</hint>
    </navigation-parameters>
    <navigation-parameters>
        <name>height</name>
        <type>String</type>
        <label xml:lang="en">height</label>
        <hint xml:lang="en">Portlet Height</hint>
    </navigation-parameters>
    <portlet-id>adf_jsf__doclib-document-library_jspx</portlet-id>
    <allow-export>true</allow-export>
    <allow-import>true</allow-import>
    <require-iframe>true</require-iframe>
    <minimum-wsrp-version>2</minimum-wsrp-version>
</portlet-extension>
```

When portlet parameters are passed from the portlet consumer, the parameters' name/value pairs are added to the porletized page's URL as request parameters. The producer can fetch these values from the URL in a backing bean. The sample workspace provides an example of a backing bean called ProducerBean that has a property called `portletSize`. The value of the property is a literal String of the format "`width: xxx; height: yyy;`" that can be used in the producer's page as a CSS `inlineStyle` attribute. The sample workspace contains the following backing bean as example, *oracle.webcenter.sample.e20producer.ProducerBean*.

Register the backing bean in faces-config.xml. For example:

```
...
<managed-bean>
  <managed-bean-name>ProducerBean</managed-bean-name>
  <managed-bean-class>
    oracle.webcenter.sample.e20producer.ProducerBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
...
```

Here is how to use an Expression Language (EL) expression to get the `portletSize` property of the ProducerBean. The return value of the EL expression is used on the portlet page in the `inlineStyle` attribute of `panelStretchLayout`. The result is that `panelStretchLayout` will be sized according to the width and height value specified in the portlet parameters. Since `panelStretchLayout` wraps around the portlet content, this means that the portlet size is constrained by the portlet width and height parameters.

```
<af:form id="f1" usesUpload="true">
  <af:panelStretchLayout id="psl1" startWidth="0px" endWidth="0px"
                         topHeight="0px" bottomHeight="0px"
                         inlineStyle="#{ProducerBean.portletSize}">
    <f:facet name="center">
    ...
```

Again, the page should be tested before consuming it as portlet to ensure it renders as intended. Geometry sizing can be tested simply by running the JSF page and providing the *width* and *height* parameters on the URL. For example:

```
http://127.0.0.1:7101/ServicesProducer/faces/doclib-document-
library.jspx?width=100%25&height=600px
```

Note: Since the width and height parameters are passed as URL parameters, the value of these parameters should be URL encoded. For example, `width=100%` should be specified as `width=100%25` where the `%` character is encoded as `%25`.

Wire WebCenter services task flow parameters to portlet parameters

WebCenter service task flows have parameters that can be used to manipulate the behavior and rendering of the task flows. In this section you will learn how to expose these task flow parameters as portlet parameters.

Declaring Portlet Parameters

The first task is to identify the list of task flow parameters to expose them as portlet parameters (in some cases it may be useful to expose only a subset for simplicity). This list of task flow parameters requires corresponding portlet parameters. The portlet parameters are declared in the producer's `WEB-INF/oracle-portlet.xml` file. The Document Library task flow, for example, can expose the `startFolderPath` task flow parameter as shown below:

```
<portlet-extension>
    <portlet-name>DocumentLibraryPortlet</portlet-name>
    <navigation-parameters>
        <name>startFolderPath</name>
        <type>String</type>
        <label xml:lang="en">Start Folder Path</label>
        <hint xml:lang="en">Start Folder Path</hint>
    </navigation-parameters>
    <navigation-parameters>
        <name>width</name>
        <type>String</type>
        <label xml:lang="en">width</label>
        <hint xml:lang="en">Portlet Width</hint>
    </navigation-parameters>
    <navigation-parameters>
        <name>height</name>
        <type>String</type>
        <label xml:lang="en">height</label>
        <hint xml:lang="en">Portlet Height</hint>
    </navigation-parameters>
    …
</portlet-extension>
```

Binding Task Flow Parameters to URL Parameters

The next task is to bind the value of the task flow parameter to the URL parameter. This can be achieved by setting the task flow parameter in the page definition. At runtime, the portlet parameters' name/value pairs are injected into the portletized page's URL as parameters by the Oracle JSF Portlet Bridge. The Document Library task flow, for example, can obtain the value for `startFolderPath` from the corresponding URL parameter using the EL expression `${param.startFolderPath}`. This expression is applied to the task flow parameter in the page definition. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel"
                version="11.1.1.52.61"
id="doclib_document_libraryPageDef"
                Package="view.pageDefs">
  <parameters/>
  <executables>
```

```
    <taskFlow id="doclibdocumentlibrary1" …>
      <parameters>
        …
        <parameter id="startFolderPath"
                   value="${param.startFolderPath}"
                   xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
      </parameters>
    </taskFlow>
  </executables>
  <bindings/>
</pageDefinition>
```

Again, the page should be tested as-is before consuming it as portlet to ensure it renders as intended. Parameters can be tested simply by running the JSF page and providing the parameter's name and value pair on the URL. For example:

```
http://127.0.0.1:7101/ServicesProducer/faces/doclib-document-
library.jspx?startFolderPath=%2Ftmp
```

Add Trinidad Style Pop-up Dialog Support

Trinidad Style Pop-up Dialog Limitations

The Oracle JSF Portlet Bridge does not currently support Trinidad-style pop-up dialogs. Trinidad-style pop-up dialogs are used to display content in a separate browser pop-up window, as opposed to inline pop-up dialogs that appear within the same browser window. Consequently, for WebCenter services task flows that use Trinidad-style pop-up dialogs, an additional effort is required.

For service task flows that do not use Trinidad-style pop-up dialogs, the tasks outlined in this section are not required. The following services task flows do not require this additional support:

- Document Library - Recent Documents
- RSS Viewer
- Search Preferences

To work around the Trinidad style pop-up dialog limitation, the producer application requires a custom implementation of a Trinidad dialog service to invoke the Trinidad dialog raised by the bridged portlets. The following tasks are required to create and register this custom Trinidad dialog service:

- Create a custom JSF phase listener to raise the dialog before the render response phase
- Create a custom Trinidad `DialogServiceImpl` to capture a request to raise the Trinidad dialog and prepare the dialog to be raised

- Create a custom Trinidad `RequestContextImpl` to instantiate a custom Trinidad `DialogServiceImpl`
- Create a custom Trinidad `RequestContextFactory` to instantiate a custom Trinidad `RequestContextImpl`
- Create and register a servlet filter to set a custom Trinidad `RequestContextFactory`

Note: The following WebCenter service task flows are currently not supported across the Oracle JSF Portlet Bridge. Capabilities of these task flows are also not supported when surfaced via other WebCenter service task flows.

- Relationship
- People Picker

Create a custom JSF phase listener to raise the dialog before the render response phase

This custom JSF phase listener contains logic to invoke the dialog window. It should be called immediately before the render response phase, and not in the `invokeApplication` phase where Trinidad typically attempts to raise pop-up dialogs. The reason the logic is invoked before the render response phase is because the `ExternalContext.encodeResourceURL()` API used in the code to raise a browser dialog across a portlet request is available only with the `renderResponse` phase (not in the `invokeApplication` phase). This logic takes as input the target URL to show in the dialog, the window width, and the window height. A Javascript snippet is constructed to invoke a browser dialog based on these parameters. The Javascript snippet is returned to the browser as part of the response and executed by the browser to raise the dialog window. The sample workspace contains the following phase listener as example, *`oracle.webcenter.sample.e20producer.JSFPhaseListener`*.

Register the phase listener in `faces-config.xml` like this:

```
<lifecycle>
  <phase-listener>
  oracle.webcenter.sample.e20producer.JSFPhaseListener
  </phase-listener>
</lifecycle>
```

Create a custom Trinidad `DialogServiceImpl` to capture requests to raise a Trinidad dialog and prepare the dialog to be raised

The existing behavior of the *`org.apache.myfaces.trinidadinternal.context.DialogServiceImpl`* must be replaced to use the custom behavior. More specifically, instead of raising the dialog right away in the `invokeApplication` phase (that's when the `launchDialog()` method is called), the URL and widow size should be retrieved and passed to the custom phase listener where the dialog is actually raised before the renderRepsone phase. These dialog-related properties are passed using the `windowProperties` Map object in the request map. The sample workspace contains the following class as example, *`oracle.webcenter.sample.e20producer.DialogServiceImpl`*.

Create a custom Trinidad RequestContextImpl to instantiate a custom Trinidad DialogServiceImpl

Trinidad *`org.apache.myfaces.trinidadinternal.context.DialogServiceImpl`* is instantiated by *`org.apache.myfaces.trinidadinternal.context.RequestContextImpl`*. Unfortunately, there is no mechanism with which to supply a custom dialog service to be instantiated instead of the out-of-the-box Trinidad *`DialogServiceImpl`*. Therefore, a custom *`RequestContextImpl`* needs to be created to instantiate a custom *`DialogServliceImpl`*. Note that the package of this class should be *`org.apache.myfaces.trinidadinternal.context`* in order for it to access members from the parent class *`org.apache.myfaces.trinidadinternal.context.RequestContextImpl`*. The sample workspace contains the following class as example, *`org.apache.myfaces.trinidadinternal.context.ServicesProducerRequestContextImpl`*.

Create a custom Trinidad RequestContextFactory to instantiate a custom Trinidad RequestContextImpl

The class *`org.apache.myfaces.trinidadinternal.context.RequestContextFactoryImpl`* currently registers Trinidad's own *`org.apache.myfaces.trinidadinternal.context.RequestContextImpl`*. To replace the existing behavior, create a custom *`RequestContextFactory`* to instantiate a custom Trinidad *`RequestContextImpl`*. The sample workspace contains the following class as example, *`oracle.webcenter.sample.e20producer.ServicesProducerRequestContextFactory`*.

Create and register a servlet filter to set a custom Trinidad RequestContextFactory

Create a servlet filter and in the init() method call the Trinidad *`RequestContextFactory`* to set the custom factory you have created. The sample workspace contains the following class as example, *`oracle.webcenter.sample.e20producer.ServicesProducerFilter`*.

Register the servlet filter in `web.xml`. Make sure this filter is placed BEFORE other filters and servlets in `web.xml`.

```
<filter>
  <filter-name>ServicesProducerFilter</filter-name>
  <filter-class>
   oracle.webcenter.sample.e20producer.ServicesProducerFilter
```

```
        </filter-class>
      </filter>
```

Workaround for Trinidad pop-up dialog when consumed by a custom WebCenter application

The following workaround is required in order for Trinidad pop-up dialogs to work across portlet bridge when consumed in WebCenter custom applications. This workaround is NOT required when consumed in Oracle Portal 11g.

1. Deploy the producer application to a WebLogic Managed Server.
2. Locate `wsrp_mime_cs_mappings.properties` in the deployed application's `WEB-INF` directory (for example, `.../system11.1.1.1.32.53.04/DefaultDomain/servers/DefaultServer/tmp/_WL_user/TestDialog_application1/896d1u/war/WEB-INF/wsrp_mime_cs_mappings.properties`)
3. Open `wsrp_mime_cs_mappings.properties` in an editor and add the following to the end of the file: `\*=ISO-8859-1`
4. Bounce the Managed Server.

Note: The approach suggested in the Add Trinidad Style Pop-up Dialog Support section does not currently fully implement the Trinidad dialog functionality. For example, the ability to pass back return values is not implemented. What is implemented is the bare-bone capability to raise a specified dialog with a specified dimension and parameters.

Service Specific Tasks

Discussions Service Task Flows

Three of the WebCenter Framework Discussion task flows require additional JSTL snippets in the page in order for the create forum/topic functionality to work correctly when portletized. These service task flows are Discussion Forums, Discussion – Sidebar View, Discussion Watched Forums. The following JSTL snippet (in bold) must be added to the top of the pages using these task flows. Without the required snippets, the create forum/topic dialog will not appear when invoked.

```
<?xml version='1.0' encoding='windows-1252'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:h="http://java.sun.com/jsf/html"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:rtc="http://xmlns.oracle.com/WebCenter/collab/rtc"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html;charset=windows-1252"/>
  <c:set value="true" var="includeCFmPupFragment" scope="session"/>
  <c:set value="true" var="includePTpPupFragment" scope="session"/>
  <f:view>
```

```
    <af:document id="d1">
    …
```

Mail Service Task Flow

The WebCenter Framework Mail task flow requires additional `web.xml` configuration for the read mail functionality to work correctly when portletized. The following `context-param` must be added to the `web.xml` of the producer application. Without this context parameter with the correct value, the pop-up window to read the selected mail will not display the body of the mail content correctly.

```
  <context-param>
    <param-name>
      oracle.adf.view.rich.security.FRAME_BUSTING
    </param-name>
    <param-value>never</param-value>
  </context-param>
```

Turning the Application into a Pagelet Producer

In order to consume the service task flows in Ensemble, the pages we've created containing the service task flows need to be wrapped with an iframe. Just like the portlets, the iframe has to be sizable according to the width and height parameters. This section shows how to create a servlet endpoint that exposes the service task flows wrapped with an iframe.

Add the Ensemble endpoint servlet filter

The first task is to create a servlet endpoint for the producer that exposes the page with the service task flows within an iframe. This servlet endpoint URL looks like this:

```
        http://<ProducerHost>:<ProducerPort>/<ProducerContextRoot>/<Servle
        tEndpointName>?id=<ServiceTaskFlowName>&width=<PageletWidth>&heigh
        t=<PageletHeight>
```

Each service endpoint will have different `serviceId` parameters. The servlet endpoint is used in defining the pagelets. Behind the servlet endpoint is a servlet filter class. The sample workspace contains the following filter class as example, *oracle.webcenter.sample.e20producer.ServicesProducerFilter*.

This servlet filter can be registered in the `web.xml` like this:

```
        <filter>
          <filter-name>ServicesProducerFilter</filter-name>
          <filter-class>
          oracle.webcenter.sample.e20producer.ServicesProducerFilter
          </filter-class>
        </filter>
        <filter-mapping>
          <filter-name>ServicesProducerFilter</filter-name>
          <url-pattern>/services</url-pattern>
        </filter-mapping>
```

The endpoint to use with Ensemble, for example the document library pagelet, could looks like this:

```
http://127.0.0.1:7101/ServicesProducer/services?id=doclib-
document-library&width=100%25&height=600px
```

Add support for pagelet geometry parameters

In the `ServicesProducerFilter` class, the width and height for the pagelet are retrieved from the request parameter and used to set the geometry of the iframe. This in turn sizes the pagelet accordingly.

Wire WebCenter services task flow parameters to URL parameters

In the `ServicesProducerFilter` class, any request parameters from the pagelet are duplicated as-is onto the iframe's source URL. Since service task flows pick up parameter values directly from the iframe's source URL, request parameters from the pagelet are passed automatically to service task flow parameters.

Configure CSP Identity Asserter

An identity assertion provider is provided with the WebCenter R1 release, which will accept the request coming from the Ensemble/Aqualogic UI (or any Web 2.0 client) and assert the identity from it. The Ensemble proxy sends the request, in the form of an encoded and zipped ALI token (the ALI token is simply a SAML 2.0 token), in the header itself. The WLS server passes this request to the CSP ID provider. The ID provider internally parses and validates the token (using OSDT API's), fetches the username and passes the identity (i.e., the username) to the Loginmodule for authentication. If an identity with the same username exists in the WLS security realm, the WLS asserts the user (i.e., identity) and thus the client's request is honored and gains access to the requested resource.

Below is the sequence of events:

1. Ensemble proxy sends request as an ALI (SAML) token
2. WLS server checks if the ID provider exists for the given token type
3. CSP ID provider asserter parses and verifies the token
4. ID provider maps the username with a WLS username
5. WLS calls the LoginModule and authenticates the user
6. Request is honored

Here are the changes required for the producer. The main change is specifying the login config to be CLIENT-CERT based. This is required so that our ID asserter is invoked.

- Ensure that the security constraint are correctly added to the resource that you want to secure. For example:

```
<security-constraint>
 <web-resource-collection>
   <web-resource-name>AuthenticatedPages</web-resource-name>
```

```
        <url-pattern>/welcomeservlet</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
        <role-name>valid-users</role-name>
      </auth-constraint>
    </security-constraint>
```

- Set login config to be client certificate based

```
    <login-config>
        <auth-method>CLIENT-CERT</auth-method>
        <realm-name/>
    </login-config>
    <security-role>
        <description>
          Any valid user with an identity store account.
        </description>
        <role-name>valid-users</role-name>
    </security-role>
```

In addition to specifying the login config, the certificate required for secured communication between the producer and Ensemble needs to be loaded into the WebLogic server.

- Obtain the CSP certificate that the Ensemble proxy uses for secure communication. This is typically done out of band, through e-mails or some other means. You can get the certificate from the Ensemble proxy administrator.
- Load the certificate into the WebLogic server trust key store using the keytool command. For example: *$JAVA_HOME/bin/keytool -import -keystore <Trust_KS_DIR>/CustTrust.jks -trustcacerts -alias WEBCENTER_CSP_ID_ASSERTER_ALIAS -file export.cert -storepass welcome1 -noprompt*
- Restart the admin server and access the WebLogic administration console as the *weblogic* administrator. For example: http://myhost.us.oracle.com:7001/console
  - Configure the Keystore as follows:
    - In WLS Console App, go to Environmet -> Server. Select the Server instance where the producer app is deployed.
    - Click on Keystore tab.
    - From the Keystore drop down select Custom and Trust Store type.
    - In the Trust section, correctly specify the Custom Trust Keystore filename and password, is any . This is the same file where you imported the client certificate in the earlier Example: <Trust_KS_DIR>/CustTrust.jks
  - Configure the CSP Id Asserter
    - In WLS Console App, go to Security Realms -> myrealm -> Providers Tab -> Authentication sub tab (first sub tab).

  - Click on New -> Give name as CSPIdentityAsserter (the name can be anything)
  - From the Type Drop Down, select CSPIdentityAsserter
  - OK you way out.
- You can control the logging by setting the properties under Logging tab for this server.
- Restart the server (both admin and managed servers if any).
- Verify that your changes have been persisted. Open domain_home/config/config.xml and it should contain the following fragment.

```
<sec:authentication-provider
  xmlns:ext="http://www.bea.com/ns/weblogic/90/security/extension"
  xsi:type="ext:csp-identity-asserterType">
 <sec:name>CSPIdentityAsserter</sec:name>
</sec:authentication-provider>
```

Note: Sometimes the changes are not saved correctly, i.e. the config.xml does not contain the xsi:type="ext:csp-...." Attribute. This is probably because of some WLS bug. If this happens, the server restart will fail. To fix this, remove the above fragment from the config.xml, restart, and redo the configuration again.

Making the Producer Session Cookie Unique

When portlets or pagelets are consumed on a portal page, the cookies set by the portlets can collide with cookies from other portlets, or even the portal depending on implementation. Therefore, it is recommended to set the producer application's cookie-path to something unique to prevent it from defaulting to "/" which often leads to cookie collisions. For example, the cookie path can be set in `weblogic.xml` like this:

```
<session-descriptor>
  <cookie-path>/WebCenterServices</cookie-path>
</session-descriptor>
```

Exposing WebCenter Services Task Flows as
WSRP Portlets and Ensemble Pagelets
August 2009
Author: Ken Young

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment