BEA Oracle®

# WebCenter Portlet Toolkit for .NET

## Development Guide

Version 10.3
Revised: October 2008

# Contents

# 3. Authoring WSRP Portlets Using the Oracle WebCenter Portlet Toolkit for .NET

# About the Oracle WebCenter Portlet Toolkit for .NET

The Oracle WebCenter Portlet Toolkit for .NET is a collection of libraries and Microsoft Visual Studio 2005 integration features that simplify authoring of ASP.NET 2.0 portlets for Oracle WebCenter Interaction and Oracle WebLogic Portal. The Oracle WebCenter Portlet Toolkit for .NET is provided with the Oracle WebCenter Application Accelerator for .NET

Each of the components in the Oracle WebCenter Portlet Toolkit for .NET serves a specific purpose:

- **.NET Portlet API**: Provides a set of project templates, file templates and a class library used to create portlets in Visual Studio. Includes separate templates and classes for WSRP (Oracle WebLogic Portal) and Oracle WebCenter Interaction portlet development. This guide is divided into two sections:
  - *#unique_2*
  - *#unique_3*
- **Web Control Consumer (WCC)**: Transforms output rendered by Oracle WebCenter Interaction portlets into pages that can be consumed in a portal environment (Oracle WebCenter Interaction only). This component is not required for WSRP portlets.

For details on configuring the Oracle WebCenter Application Accelerator for .NET, see the *Installation Guide for Oracle WebCenter Application Accelerator for .NET*.

# Additional Documentation

The following additional documentation is available.

| Resource | Description |
|---|---|
| Installation Guide | This guide describes the prerequisites (such as required software) and procedures for installing and upgrading Oracle WebCenter Portlet Toolkit for .NET components. |
| Release Notes | The release notes provide information about new features, issues addressed, and known issues in the release. |
| Development Guide | The *Oracle WebCenter Portlet Toolkit for .NET Development Guide* provides detailed information on authoring portlets using the .NET Portlet Toolkit. |

CHAPTER 2

# Authoring Portlets for Oracle WebCenter Interaction Using the Oracle WebCenter Portlet Toolkit for .NET

The Oracle WebCenter Portlet Toolkit for .NET provides ease-of-use features for authoring Oracle WebCenter Interaction portlets. All Oracle WebCenter Interaction Development Kit (IDK) features are accessible through the Oracle WebCenter Portlet Toolkit for .NET, and some are much simpler to use, requiring only a single line of code to accomplish complex tasks.

This section contains the following topics:



The Oracle WebCenter Portlet Toolkit for .NET is fully integrated with Microsoft Visual Studio 2005 and includes pre-configured project templates, file templates for portlet and preference pages, and a

class library that provides easy access to preferences, user properties, profile information, and other useful information. To deploy portlets developed with the .NET Portlet Toolkit in Oracle WebCenter Interaction, follow the normal steps for configuring the associated objects. For details on deploying portlets in Oracle WebCenter Interaction, see the *Oracle WebCenter Interaction Web Service Development Guide*.

# Creating an Oracle WebCenter Interaction Portlet Using the .NET Portlet Toolkit

To create a new Oracle WebCenter Interaction portlet, use the Microsoft Visual Studio 2005 templates provided with the .NET Portlet Toolkit. The templates are pre-configured to include references to required assemblies, including the Oracle WebCenter Interaction Development Kit (IDK), Portlet API libraries, the Logging Spy infrastructure and the Oracle WebCenter Interaction Development Kit (IDK) user profile provider.

A new portlet project includes a simple portlet page in the Default.aspx/Default.aspx.cs files. This page extends the `ALIPortletPage` base class, which provides access to preferences, user information and IDK resources such as the PortletRequest and PortletResponse.

Follow the steps below to create a new portlet using the `ALIPortletPage` template.

1. Create a new Oracle WebCenter Interaction portlet project in Microsoft Visual Studio 2005:
   a) To create a new project, click **File ➤ New ➤ Web Site** .
   b) Select the language option: C#.
   c) In the My Templates list, select ALI Portlet Project.
   d) Enter a name for the project (for example, "simpleWeb").
2. Create a new portlet page:
   a) In the **Solution Explorer**, right-click on the root of the project and select Add New Item….
   b) In the **My Templates** list, select ALI Portlet Page.
3. Add the necessary functionality to the portlet page:
   a) *Manipulating Oracle WebCenter Interaction Preferences Using the .NET Portlet Toolkit* on page 9
   b) *Accessing User Profile Information Using the .NET Portlet Toolkit* on page 10
   c) *Accessing Additional Portlet Information Using the .NET Portlet Toolkit* on page 11

# Manipulating Oracle WebCenter Interaction Preferences Using the .NET Portlet Toolkit

To access and update Oracle WebCenter Interaction preferences using the .NET Portlet Toolkit, use preference attributes or a preference collection.

**Preference attributes** provide type specific read-write binding of an Oracle WebCenter Interaction preference to a page member variable (field). One of the following access modifiers is required on all local fields that are bound to a preference using a preference attribute: [protected | internal | protected internal | public]. The modifier "private" is not supported. Since this is the default modifier for a field if none is specified, the modifier must not be omitted. Preference attributes can be bound to fields with types of String, int, float, bool, or DateTime. Values for fields marked with preference attributes are set after the OnLoad() page event; any changes are persisted during the OnUnload() page event.

`ALIPortletPage` exposes a preference attribute for each type of preference: `PortletPreferenceAttribute`, `CommunityPreferenceAttribute`, `CommunityPortletPreferenceAttribute`, `AdminPreferenceAttribute`, `UserPreferenceAttribute`, and `SessionPreferenceAttribute`. All preference attributes have two optional properties:

- Key: A string that provides the key name of the property to which the member variable is bound. If you do not include this property, the name of the preference defaults to the name of the variable. Key values must be unique within a page.
- DefaultValue: A string that provides the value to use when the preference is unavailable or has not yet been set.

**Note:** Private variables (including those with no permissions modifier) cannot be seen by reflection and are not supported. You must include a permissions modifier of `public`, `protected` or `internal`.

```
[PortletPreference]
protected int myIntPref;

[PortletPreference(Key = "lastAccessed", DefaultValue =
"2008-01-01")]
protected DateTime lastAccessedDateTime;
```

**Preference collections** allow you to enumerate all of the preference values in the request using the PreferenceCollection.Keys property. `ALIPortletPage` provides a preference collection for each type of preference: `PortletPreferences`, `CommunityPreferences`, `CommunityPortletPreferences`, `AdminPreferences`, `UserPreferences`, and `SessionPreferences`.

```
String myPrefValue = this.PortletPreferences["MyPrefName"];
this.PortletPreferences["MyPrefName"] = myPrefValue;
```

> The examples above use `PortletAttribute` and `PortletPreferences`, but could be replaced with Admin*, User*, Community*, CommunityPortlet* or Session* to read preferences of the corresponding type. (Most preferences can also be set from a portlet page, but Admin preferences must be set from an preference page.)

For a full description of the API, see the complete class documentation for the Oracle WebCenter Interaction ("ALI") section of the Portlet API, installed into the Microsoft Visual Studio 2005 Integrated Help System.

For details on the types of preferences available in Oracle WebCenter Interaction, see the *Oracle WebCenter Interaction Web Service Development Guide*.

# Accessing User Profile Information Using the .NET Portlet Toolkit

To access user profile information using the .NET Portlet Toolkit, add the properties to the Web.config file and reference them by name in the portlet page.

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The .NET Portlet Toolkit Portlet API uses the ProfilerProvider API to expose user profile information sent by Oracle WebCenter Interaction. For more information about the ProfileProvider model, see the *MSDN documentation*.

The Web.config file for a .NET portlet project contains a `<profile>` element that defines the user profile properties supported by a Provider. In order for user profile properties from the source application to be accessible from the ASP.NET Profile object, they must be registered in the Web.config file.

**Note:** User Profile information is read-only and can only be modified through the source application.

1. Add the user profile properties to a `<properties>` element within the `<profile>` element in the Web.config file for the portlet project. Each property must be defined with a name and a type. If you provide a default value, it will be used if the property is not available.

   **Note:** The name of the property **must** match the name sent by the source application.

   ```
   <properties>
   <add name="CustomPropertiesTitle" type="string" defaultValue="No
    Title"/>
   <add name="MyProfileName" type="String" defaultValue="Guest"/>
   <add name="MyProfileAge" type="Int32"/>
   </properties>
   ```

2. Reference the property in the portlet page by name using the Profile object.

   ```
   string title = Profile.CustomPropertiesTitle
   string name = Profile.MyProfileName;
   int age = Profile.MyProfileAge;
   ```

# Accessing Additional Portlet Information Using the .NET Portlet Toolkit

To access portlet-specific information from Oracle WebCenter Interaction, use the .NET Portlet Toolkit objects in the `ALIPortletPage` base class.

In addition to preferences and user profile information, a variety of other portlet-specific information is available from a portlet page that extends `ALIPortletPage`. This page provides examples. For a complete list, see the API documentation.

- To access the current portal session (IRemoteSession), use the following code:

  ```
  IRemoteSession session = this.RemotePortalSession;
  ```

- To access the current user's name from Oracle WebCenter Interaction, use the following code:

  ```
  String name = User.Name;
  ```

- To access the host page URI, use the following code:

```
URI hostPage = HostPageURI;
```

# Creating Oracle WebCenter Interaction Preference Pages Using the .NET Portlet Toolkit

To create a preference page, use the VisualStudio.NET templates provided with the .NET Portlet Toolkit. The templates are pre-configured to include references to required assemblies, including the Oracle WebCenter Interaction Development Kit (IDK), Portlet API libraries, the Logging Spy infrastructure and the Oracle WebCenter Interaction Development Kit (IDK) user profile provider.

The ALIPreferencePage template provides the same functionality as an ALIPortletPage and includes additional features for rendering and editing a preference page. By default, the page renders a title bar with "Finish" and "Cancel" buttons that match the standard ALI preference page style. The appearance and functionality of the title bar can be changed by overriding the `DrawTitleBar()` method from the base class.

To complete the preference page, you must implement four methods:

| Method Name | Description |
|---|---|
| `GetTitleBarText()` | Read-only property to get the text for the title bar. |
| `GetFinishButtonText()` | Read-only property to get the text for the finish button. |
| `GetCancelButtonText()` | Read-only property to get the text for the cancel button. |
| `SavePreferences()` | Method implemented to update preference values for any preferences changed on the page. |

# Logging .NET Application Accelerator Activity

To view logs of .NET Application Accelerator activity, use Oracle WebCenter Interaction Logging Spy.

Oracle WebCenter Interaction Logging Spy is automatically installed with Oracle WebCenter Interaction or installed on a remote server as part of the Oracle WebCenter Logging Utilities, a stand-alone package. Logging Spy listens for log messages transmitted using network broadcast. By configuring a list of message senders, you can listen in on log traffic from multiple Oracle WebCenter applications in a single console.

The .NET Application Accelerator includes four logging message senders:

| Logging Message Sender | Description |
|---|---|
| .NET WSRP Producer | The stand-alone web application designed to serve as a WSRP producer for Oracle WebLogic Portal. This name is not configurable and transmits all messages related to the WSRP Producer. The WSRP producer "loggingName" entry in Web.config is not currently used. |
| ALIPortletProject | Transmits log messages from the portlet ASP.NET web application. The name of this message sender is defined in the Web.config file of any ASP.NET portlet project built using the .NET Portlet Toolkit. The name of the sender defaults to the project name entered when the project was created, but can be changed by modifying the value attribute of the following entry:<br>`<add key="ptedk.LoggingApplicationName" value="ALIPortletProject"/>` |
| BEA Portlet API | Transmits log messages from the .NET Portlet API. |
| ALI .NET Application Accelerator | Reserved for future use. No logging traffic is sent by this sender. |

To add a .NET Application Accelerator message sender to your Spy configuration, follow the instructions below.

1. Open Logging Spy.
2. Select **View ➤ Set Filters** .

3. In the **Filter Settings** dialog, click **Edit ➤ Add Message Sender** and choose a message sender (see the table above).

For more information on configuring Logging Spy, see the *Oracle WebCenter Interaction Web Service Development Guide* and the *Administrator Guide for Oracle WebCenter Interaction*.

# About the .NET Web Control Consumer (WCC)

The .NET Web Control Consumer provides support for using standard .NET Web Controls in portlet development.

Two problems arise with using .NET Web Controls out of the box for portlet development:

- Client-side scripting problems due to naming collisions: .NET assumes it occupies the whole page, and all specific functions are always named the same. These problems occur primarily with validator controls. Validator controls define a page scope variable (`Page_Validators`), a page scope function (`ValidatorOnSubmit()`), and also write some inline JavaScript code.
- Postback problems cause naming collisions in addition to posting back to the incorrect page: The preferred behavior for Web Control is generally an in-place refresh. For example, when a user selects the next month in the Calendar control, it posts back to the originating aspx page. Even though the post goes through the gateway, it is not sent to the portal page that hosted it, and the user loses the portal experience.

The .NET Web Control Consumer package overcomes these problems and provides a simplified experience for portlet developers. The core of the package is a .NET HttpModule that modifies the outgoing HTML before it reaches the portal server. The module solves the problems described above by: a) individuating all .NET provided function names to avoid collisions, and b) modifying all postback functions to use JavaScript instead of posting directly to the server.

# Configuring Oracle WebCenter Interaction Portlets in the Web Control Consumer

To configure an existing .NET Web application to be used in Oracle WebCenter Interaction, you must make a new entry in the web application's Web.config file.

Once you add an entry to the Web.config file, all gatewayed requests to the web application are transformed for use with Oracle WebCenter Interaction. You can still access your web application directly; the module is bypassed for any request that does not originate from a portal server. For example templates, see the following folders in the installation directory: \deploy\Web.config and \deploy\web.config.node (node only). Your IIS Web Application Server must have direct web access to the portal Image Service.

**Note:** Any .NET portlet page that performs postbacks (including auto-postback) must be gatewayed. For details on configuring the gateway, see the *Oracle WebCenter Interaction Web Service Development Guide* or the portal online help.

- In the /configuration/system.web node of the Web.config file, add a new node that specifies the HttpModule class and containing assembly, as shown in the example below:

```
<httpModules>
 <add type="Com.Plumtree.Remote.Loader.TransformerProxy,
Aqualogic.WCLoader, Version=3.1.0.0, Culture=neutral,
PublicKeyToken=d0e882dd51ca12c5" name="PTWCFilterHttpModule"
/>
</httpModules>
```

# Configuring Image Server Access

If the URL used to access the portal Image Service internally is different from the URL used for external access, and the portlet server is hosted internally, you must configure the Image Service address.

Some network configurations require that the portal Image Service be accessed through a different URL internally and externally. This can be a problem for portlet servers that are hosted internally because Oracle WebCenter Interaction always sends the external Image Service URL (the portlet must contact the Image Service to retrieve auxiliary JavaScript files). If this is the case, set up a mapping to allow the portlet server to determine the internal Image Service address. This can be achieved in two ways.

The simplest way is to map the external image server to the internal URL by following the steps below:

1. Open the imageserver.mapping.xml file in a text editor. This file is located in the .NET Web Controls installation folder, under \settings\config.

2. Add a new entry for the internal Image Service, as shown below:

   ```
   <mapping find="http://www.external-servername.com/ptimages/"
   replace="http://internal-servername:8080/ptimages/"/>
   ```

   You must include the whole URL; sections of the URL will not be replaced. To replace sections, you must use a regular expression mapper, as shown below:

   ```
   <mapping regex="true" find="www.external-(\w+).com"
   replace="internal-$1:8080" />
   ```

   (Both mappings do the same thing in this case. For details on syntax, see .NET regular expressions.)

3. Save the imageserver.mapping.xml file.
4. Restart IIS or re-save the config.xml file located in the same directory (this instructs the Web Controls to reload all configuration settings).

The second option is to set an alternative Image Service URL to override the external URL, using an Administrative preference. The default setting name is `PTWC.Mapping.Override` (this name can be changed in the HttpPipe.xml file). You must create an administrative preferences page to set the preference. This granularity of configuration is not generally necessary except for specific remote portlets that access the Image Service through a specific unique URL.

# Configuring .NET Framework Support for the Web Control Consumer

To add support for any .NET Framework version that uses JavaScript WebUIValidation, use the JSGenerator application.

The .NET Web Control Consumer is shipped with support for a few explicit versions of the .NET Framework. You can add support for any .NET Framework version that uses JavaScript.WebUIValidation (version 125) using the JSGenerator application. To generate a new version of JavaScript files, follow the steps below:

1. Start the JSGenerator application by running JSGenerator.exe located in the \bin directory of the WCC installation directory.
2. Select the new Framework version from the list, or enter the version in the format X.X.X.X. (The list includes all versions available on the local machine.)
3. Click Generate.
4. Copy the generated files. (Once generation is complete, the application will provide instructions on where to copy the new files to the Image Service.)

Once you have copied the files, the new version of the .NET Framework will be supported.

# .NET Web Control Consumer Development Best Practices

The following tips and best practices should be considered for all portlets that use the .NET Web Control Consumer.

Follow the basic guidelines below in any portlet that uses the .NET Web Control Consumer:

- Name all elements uniquely.
- Do not assume your code is in a particular location in the HTML DOM.
- Do not rely on back/forward buttons or JavaScript commands.
- Do not access validator spans directly.
- Use FlowLayout instead of GridLayout.
- Always include the target page in the gateway space.

For more information and additional best practices, see the sections that follow.

### Using PostBacks

The .NET Web Control Consumer module enables the use of .NET Web Controls and their associated AutoPostBack function. All automatic postbacks are caught and used to repaint the individual portlet. The module makes a number of modifications to the HTML; below is an overview of the modifications you should be aware of:

- All POSTs are handled by JavaScript, so they do not become part of the browser history. This means that JavaScript functions will not work; the Back and Forward buttons of the browser will skip any clicks made on Web Controls that are handled in this way.
- All requests from a portal server are transformed; navigating to the portlet directly does not yield the same HTML that is returned to the portal server. This allows the ASPX page to be accessed directly if required (the modified page would not function outside the portal environment). To examine the HTML going to the portal server, use a trace tool between the portal and the portlet server to intercept the HTML.
- All client-side validation script is transformed, but since ASP.NET only generates client-side script for specific versions of Internet Explorer (IE), nothing is altered for browsers other than IE.
- All .NET portlets are wrapped in an HTML `<span>`. Do not assume that the portlet is at any specific point in the HTML structure of the page; portlet parent elements are usually HTML table cells, but not in this case.

- The `_doPostBack` method is removed (a similar function is included in an additional JavaScript file). Beware of hooking into this .NET-provided function directly; both the function name and argument list are modified. Any direct calls to this function should also be modified automatically, however any non-standard calling conventions could potentially not be recognized by the module. To programmatically perform a postback from JavaScript, make the call `_doPostBack('','');`.
- Always use FlowLayout because GridLayout uses absolute positioning.
- The names for HTML spans (for validators) and forms (required for all pages that perform a postback) are appended with a unique ID (portlet ID), to avoid naming collisions on a portal page. Do not refer to these names directly in JavaScript. (If it is absolutely required, you can assume that they will be named <original-name>_<portlet-id>.)

### Using Unique Naming

Generally all HTML elements should be named uniquely, which normally involves appending the portlet ID to the element name. This can be problematic with ASP.NET, because the Web Control names and IDs cannot be generated at runtime in the normal ASPX syntax. For example the following code will not work:

```
<asp:Button id="Execute_<%= portlet_id %>" runat="server"
Text="Test"></asp:Button>
```

The best way to append the portlet ID to the name is programmatically. To do this, declare the control in the ASPX page with a standard name, and then modify the ID property from the code behind page. You can access the actual controls using the `System.Web.UI.Page Controls` property or the `System.Web.UI.Page FindControl(string ControlName)` method.

For example, to append the portlet ID to a control named "MyButton," the following code could be employed:

```
protected override void OnPreRender(EventArgs e)
{
 base.OnPreRender (e);
 FindControl("MyControl").ID += portletID;
}
```

**Note:** It is simplest to append the ID using the page's PreRender event so that the standard ID is used until the last possible point.

### Using Submit Buttons

In most cases, submit buttons will be modified to remain in the portal context. This defaults to true, although it can be disabled by adding a `ptrender=false` attribute to the button.

**Note:** It is possible to place a `ptrender` tag on the form itself. This means that anything attempting to submit the form will cause it to be posted back in-page. However, we recommend using the tag on the buttons, unless it is specifically required (for example, a third-party control is attempting to submit the form directly).

The standard method of posting to the target page and redirecting back to the portal page using the IDK is also possible, however this is generally slower (performs a post and a redirect), less aesthetically pleasing (refreshes the whole page), and will lose the state of the page if handled by the client (__VIEWSTATE will be lost).

### Using Custom JavaScript

Each page refresh dynamically writes the HTML to re-render the portlet, so custom JavaScript can pose a problem. Because any JavaScript is now executing as the portlet is rendering, there are some functions that will not work correctly. Any `document.write(..)` calls will not work. To call a script each time the page is refreshed (or every time the portlet is re-rendered), use the rerenderPortletID event in the ALI Scripting Framework. This event is available only to .NET portlets and is named using the current portlet ID. For example, to create a JavaScript alert each time the portlet is rendered, use the code below.

**Note:** JavaScript is executed for each postback; make sure that the script to register for the event is only included in the first page; otherwise the function will be registered multiple times.

```
<pt:namespace pt:token="$$PORTLET_ID$$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
function alertMe_$$PORTLET_ID$$, () {
      alert("Portlet Rendering!");
}
alertMe_$$PORTLET_ID$$;//call the first time the portlet is loaded
```

Then register for the event in the code behind page:

```
private void Page_Load(object sender, System.EventArgs e)
{
 if(!IsPostBack)
  RegisterClientScriptBlock("rerender", "<script
language=\"javascript\">document.PCC.RegisterForWindowEvent('rerender.$$PORTLET_ID$$',
 alertMe_$$PORTLET_ID$$)</script>");
}
```

In the example above, $$PORTLET_ID$$ is replaced by the portlet ID using the `pt:token` tag. You can also use the Oracle WebCenter Interaction Development Kit (IDK) to extract the portlet ID from the request. See the *Oracle WebCenter Interaction Web Service Development Guide* for more information on adaptive tags.

### Disabling Filters

Under some circumstances, it may be desirable to disable HTML filtering. Pages opening in popup windows or operating in gatewayed mode will probably not want any HTML modification. Filtering can be disabled per-request by making the following call:

```
Context.Items["PTWC:EnableFilter"] = false;
```

This will disable filtering for the current request only.

## .NET Web Control Consumer Frequently Asked Questions

The following questions address common issues in portlet development using the .NET Web Control Consumer (WCC).

1. **Where are the WCC web controls in VS.NET?** The WCC does not provide any additional ASP.NET controls; it offers support for the existing ASP.NET controls available with VS.NET to operate as a portlet within a portal enviroment.
2. **Which dlls do I need to add as references to my project in Visual Studio?** None. All the assembly loading is handled automatically when the HttpModule line is included in your Web.config file.
3. **Do I need to use the `ReturnToPortal` call to go back to the portal page?** Usually, no. If your portlet performs all its logic on the main portal page, then you will never leave the page, so you don't need to return to it. The exception is if you use gatewayed or hosted mode, in which case you can use this call to return to the aggregated page.
4. **Why was the stylesheet link replaced with some JavaScript?** The stylesheet must be appended to the main HTML DOM programmatically, otherwise it will be unloaded upon postback and omitted upon refresh. Instead of including your stylesheets in the standard way (`<link type="text/css" rel="stylesheet" href="http://portal-img.plumtree.com/ptimages/plumtree/common/public/css/mainstyle-en.css"/>`), the filter rewrites the link and appends it to the page using JavaScript.
5. **How do I call a custom JavaScript function as soon as the portlet refreshes itself?** Call the function from an inline piece of script.
6. **How do I run a custom JavaScript function as soon as another portlet refreshes itself?** Register the function for the portlet's rerender event. This can be done with the following JavaScript call, where $$PORTLET_ID$$ is the ID for the portlet for which to listen: `document.PCC.RegisterForEvent(document.PCC.WindowEventURN, "rerender.$$PORTLET_ID$$", myCustomFunction);`
7. **How do I make my portlet refresh periodically to keep its data up to date?** Call the postback function from a timeout. For example, the code below specifies a 10msec timeout: `<script`

```
language="javascript">setTimeout(10,
"__doPostBack('','')");<script>
```

8. **Why can't I upload files from a form?** File upload cannot be done within the page, so in-page refresh is disabled for any multi-part forms. For these forms, you must redirect back to the aggregated page manually or perform the upload in a popup window.
9. **How do I disable the filter on a specific page?** Add the following call to your code to disable the filter for the current request: `Context.Items["PTWC:EnableFilter"] = false;`

CHAPTER 3

# Authoring WSRP Portlets Using the Oracle WebCenter Portlet Toolkit for .NET

The Oracle WebCenter Portlet Toolkit for .NET provides ease-of-use features for authoring WSRP portlets. The .NET Portlet Toolkit integrates with Microsoft Visual Studio 2005 providing pre-configured project templates, a portlet item template, and a class library that provides easy access to portlet properties, user properties, user profile information, registration information, and more.

This section contains the following topics:

WSRP (Web Services for Remote Portlets) is a W3C standard for consuming one or more remote markup "Producers" from a markup "Consumer". The Oracle WebCenter Application Accelerator for .NET includes a WSRP Producer for ASP.NET that can be used to produce portlets for any WSRP Consumer, including Oracle WebLogic Portal.

# Creating a WSRP Portlet Using the .NET Portlet Toolkit

To create a new WSRP portlet, use the Microsoft Visual Studio 2005 templates provided with the Oracle WebCenter Portlet Toolkit for .NET. The templates are pre-configured to include references to required assemblies, including the .NET Portlet API.

A new portlet project includes a simple portlet page in the Default.aspx/Default.aspx.cs files. This page includes the namespaces necessary for accessing the WSRP Portlet API classes such as `PortletPropertyAttribute` and the `WSRPPortletContext`.

Follow the steps below to create a new portlet using the `WSRPPortletPage` template.

1. Create a new WSRP portlet project in Microsoft Visual Studio 2005. Note: Oracle WebCenter Interaction ("ALI") Portlet and Preference Page templates cannot be used for WSRP portlets.
   a) To create a new web application project, follow the instructions below:
      1. Click **File ➤ New Project**
      2. Under **Project types**, select Visual C#.
      3. In the **My Templates** list, select WSRP Portlet Web Application.
      4. Enter a name for the project.

   b) To create a new Web Site project, follow the instructions below:
      1. Click **File ➤ New ➤ Web Site** .
      2. Select the language option: C#.
      3. In the **My Templates** list, select WSRP Portlet Project. (Oracle WebCenter Interaction ("ALI") Portlet and Preference Page templates cannot be used for WSRP portlets.)
      4. Enter a name for the project.

2. Create a new portlet page:
   a) In the **Solution Explorer**, right-click on the root of the project and select Add New Item….
   b) In the **My Templates** list, select WSRP Portlet Page.
   c) Save the page with an intuitive name (for example, “HelloWorld.aspx”).
3. Add the necessary functionality to the portlet page:
   - *Manipulating WSRP Properties Using the .NET Portlet ToolkitTo use WSRP properties in a portlet, configure the properties in the WSRP Producer and use the .NET Portlet API. To access and define properties, you can use a property attribute or a property collection.*
   - *Accessing User Profile Information Using the .NET Portlet Toolkit* on page 27
   - *Implementing WSRP Portlet Modes Using the .NET Portlet Toolkit* on page 29
   - *Implementing WSRP Portlet Window States Using the .NET Portlet Toolkit* on page 32
   - *Managing SSO Authentication with Oracle WebLogic PortalTo manage single-sign on (SSO) authentication with Oracle WebLogic Portal, configure both Oracle WebLogic Portal and the Oracle WebCenter Application Accelerator for .NET to use either SAML or UNT authentication.*

   If the portlet should support CSS styling applied by the WSRP Consumer , you must use the CSS class names defined in the WSRP specification.
4. Register the portlet with the WSRP Producer. For details, see *Registering a Portlet with the WSRP Producer* on page 50.
5. Deploy the portlet in WLP. For details, see*Consuming a .NET WSRP Portlet in Oracle WebLogic Portal* on page 39.

# Accessing User Profile Information Using the .NET Portlet Toolkit

To access user profile information using the .NET Portlet Toolkit, add the properties to the Web.config file and reference them by name in the portlet page.

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The .NET Portlet Toolkit Portlet API uses the ProfilerProvider API to expose user profile information sent by a WSRP Consumer. For more information about the ProfileProvider model, see the *MSDN documentation*.

The Web.config file for a .NET portlet project contains a <profile> element that defines the user profile properties supported by a Provider. In order for user profile properties from the source application to be accessible from the ASP.NET Profile object, they must be registered in the Web.config file.

**Note:** User Profile information is read-only and can only be modified through the source application.

1. For each custom property set or user profile grouping, create a <group> element in the <properties> element within the <profile> element in the Web.config file for the portlet project. Add each user profile property to its respective group. Each property must be defined with a name and a type. If you provide a default value, it will be used if the property is not available.

   **Note:** The name of the property **must** match the name sent by the source application. For more information about sending user profile properties from Oracle WebLogic Portal, see *Developing User Profiles* in the Oracle WebLogic Portal documentation.

   ```
   <properties>
    <group name="homeInfo">
     <group name="online">
      <add name="email" type="string"/>
     </group>
    </group>
    <group name="CustomProperties ">
     <add name="title" type="string" defaultValue="No Title"/>
    </group>
    <group name="MyProfile">
     <add name="Name" type="String" defaultValue="Guest"/>
     <add name="Age" type="Int32"/>
    </group>
   </properties>
   ```

2. Reference the property in the portlet page by name using the Profile object.

   ```
   string title = Profile.CustomProperties.Title
   string name = Profile.MyProfile.Name;
   int age = Profile.MyProfile.Age;
   ```

3. Configure the .NET WSRP Producer to request user profile properties from the WSRP Consumer. Add a <user-profile> element to the end of the corresponding <portlet> element and configure the property names that should be passed to the WSRP Producer as shown in the example below. If you are using Oracle WebLogic Portal, the properties from a property set can be filtered individually using a string with the format

<propertyset-name>/<propertyname>. Using "*" for the <propertyname> will request all properties from a property set from Oracle WebLogic Portal. (If you are using another WSRP Consumer, consult the related documentation for the correct syntax for user profile properties.)

```
<user-profile>
  <item>CustomProperties/Title</item>
  <item>MyProfile/*</item>
</user-profile>
```

# Accessing User Profile Information Using the .NET Portlet Toolkit

To access user profile information using the .NET Portlet Toolkit, add the properties to the Web.config file and reference them by name in the portlet page.

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The .NET Portlet Toolkit Portlet API uses the ProfilerProvider API to expose user profile information sent by a WSRP Consumer. For more information about the ProfileProvider model, see the *MSDN documentation*.

The Web.config file for a .NET portlet project contains a <profile> element that defines the user profile properties supported by a Provider. In order for user profile properties from the source application to be accessible from the ASP.NET Profile object, they must be registered in the Web.config file.

**Note:** User Profile information is read-only and can only be modified through the source application.

1. For each custom property set or user profile grouping, create a <group> element in the <properties> element within the <profile> element in the Web.config file for the portlet project. Add each user profile property to its respective group. Each property must be defined with a name and a type. If you provide a default value, it will be used if the property is not available.

**Note:** The name of the property **must** match the name sent by the source application. For more information about sending user profile properties from Oracle WebLogic Portal, see *Developing User Profiles* in the Oracle WebLogic Portal documentation.

```
<properties>
 <group name="homeInfo">
  <group name="online">
   <add name="email" type="string"/>
  </group>
 </group>
 <group name="CustomProperties ">
  <add name="title" type="string" defaultValue="No Title"/>
 </group>
 <group name="MyProfile">
  <add name="Name" type="String" defaultValue="Guest"/>
  <add name="Age" type="Int32"/>
 </group>
</properties>
```

2. Reference the property in the portlet page by name using the Profile object.

```
string title = Profile.CustomProperties.Title
string name = Profile.MyProfile.Name;
int age = Profile.MyProfile.Age;
```

3. Configure the .NET WSRP Producer to request user profile properties from the WSRP Consumer. Add a `<user-profile>` element to the end of the corresponding `<portlet>` element and configure the property names that should be passed to the WSRP Producer as shown in the example below. If you are using Oracle WebLogic Portal, the properties from a property set can be filtered individually using a string with the format `<propertyset-name>/<propertyname>`. Using "*" for the `<propertyname>` will request all properties from a property set from Oracle WebLogic Portal. (If you are using another WSRP Consumer, consult the related documentation for the correct syntax for user profile properties.)

```
<user-profile>
  <item>CustomProperties/Title</item>
  <item>MyProfile/*</item>
</user-profile>
```

# Implementing WSRP Portlet Modes Using the .NET Portlet Toolkit

To access or update portlet mode for a WSRP portlet, use the `PortletInfo.Mode` property in the .NET Portlet API.

WSRP portlets can expose functionality in different modes, with each mode catering to a particular function. Many WSRP Consumers provide functionality to let users request portlet markup in various modes. There are 4 "standard" modes defined in the WSRP 1.0 specification:

- wsrp:view
- wsrp:edit
- wsrp:help
- wsrp:preview

WSRP-compliant portlets must support the wsrp:view mode. Producers can also support custom modes, defined as URIs.

Two steps are required to implement portlet modes:

1. Configure the supported portlet modes in the wsrp-producer.xml file.

   The supported portlet modes for each portlet are specified in the wsrp-producer.xml configuration file as children of the `<supports>` element as shown in the example below. For a custom mode, the description may either be included in-line in the `<supports>` element or as part of a top-level `<custom-portlet-mode>` definition.

   Each supported mode must have an associated URL. Either include the URL for the mode within the `<url>` node, or use the `idref` attribute to refer to a URL defined as a child of the `<portlet>` element. If neither an URL nor the idref is specified, the URL is assumed to be the first one found for the portlet.

   If the `<supports>` element contains multiple mime-type elements, the portlet mode support applies to each of the mime-types included. To specify different portlet modes on a per mime-type basis, add multiple `<supports>` elements to the portlet definition, each with a different mime-type. If the same mime-type is listed more than once, an exception is thrown.

**Note:** If the <portlet-mode> element is omitted, the portlet is assumed to support the wsrp:view portlet mode. If wsrp:* is specified, it implies all of the standard wsrp: values. Support for "wsrp:view" is always assumed, even if it is not explicitly indicated.

```
<portlet>
 ...
 <url
id="default">http://localhost:4614/WSRPSamples/StockQuotePortlet.aspx</url>

 ...
 <supports>
  <mime-type>text/html</mime-type>
  <portlet-mode>
   <name>wsrp:view</name>
   <url idref="default"/>
  </portlet-mode>

  <portlet-mode>
   <name>wsrp:edit</name>
   <url idref="default"/>
  </portlet-mode>

  <portlet-mode>
   <name>wsrp:help</name>
   <url idref="default"/>
  </portlet-mode>

  <portlet-mode>
   <name>wsrp:preview</name>

<url>http://localhost:4614/WSRPSamples/StockQuotePortlet_preview.aspx</url>

  </portlet-mode>

  <portlet-mode>
   <name>urn-myproject-portletmodes:search</name>
    <url>http://myhost/myportlet/search.aspx</url>
    <description lang="en">shows the search view for the
portlet</window-state>
  </portlet-mode>
  ...
 </supports>
...
<custom-items>
 <custom-portlet-mode>
```

```
  <name>urn-myproject-portletmodes:search</name>
  <description lang="en">shows the search view for the
portlet</description>
 </custom-portlet-mode>
</custom-items>
...
</portlet>
```

2. Implement the portlet mode in the portlet code.

   A portlet can access the portlet mode via the `PortletInfo.Mode` property during any portlet request, but may only change the current mode during a `performBlockingInteraction` (typically, an ASP.NET form postback). The new portlet mode is returned in the response to the WSRP Consumer. If the WSRP Consumer does not understand the mode returned by a portlet, it can ignore it. If the WSRP Consumer requests a mode that the portlet does not understand, the WSRP Producer will map to wsrp:view.

   The example below toggles the display of panels based on the portlet mode. This code is taken from the StockQuotePortlet.aspx.cs sample included with the Oracle WebLogic Portal Application Accelerator for .NET.

```
...
protected void Page_PreRender(object sender, EventArgs e)
    {
        if (WSRPPortletContext.Current.Portlet.Mode ==
"wsrp:edit")
        {
            editPanel.Visible = true;
            helpPanel.Visible = false;
            // set the stockSymbols textbox to the current
property value
            stockSymbolsTextBox.Text = stockSymbols;
            viewPanel.Visible = false;
        }
        else if (WSRPPortletContext.Current.Portlet.Mode ==
"wsrp:help")
        {
            editPanel.Visible = false;
            helpPanel.Visible = true;
            viewPanel.Visible = false;
        }
        else // view mode
        {
            editPanel.Visible = false;
            helpPanel.Visible = false;
```

```
                viewPanel.Visible = true;
                // display the quotes
                DisplayQuotes(stockSymbols);
            }
        }
...
```

# Implementing WSRP Portlet Window States Using the .NET Portlet Toolkit

To access or update window state for a WSRP portlet, use the `PortletInfo.WindowState` property in the .NET Portlet API.

Window state defines how much markup a portlet should generate. There are 4 "standard" window states defined in the WSRP 1.0 specification:

- wsrp:normal
- wsrp:minimized
- wsrp:maximized
- wsrp:solo

In Oracle WebLogic Portal and Oracle WebCenter Interaction, these window states are implemented by changing the screen real estate provided to the portlet. All WSRP-compliant portlets must support the wsrp:normal window state. To support additional window states, configure the portlet as described below. Producers can also support "custom" window states, defined as a URI.

Two steps are required to implement multiple window states in a portlet:

1. Configure the supported window states in the wsrp-producer.xml file.

   The supported window states for each portlet are specified in the wsrp-producer.xml configuration file as children of the `<supports>` element as shown in the example below. For a custom window state, the description may either be included in-line in the `<supports>` element or as part of a top-level `<custom-window-state>` definition.

   If the `<supports>` element contains multiple mime-type elements, the window state support applies to each of the mime-types included. To specify different window states on a per

mime-type basis, add multiple <supports> elements to the portlet definition, each with a different mime-type. If the same mime-type is listed more than once, an exception is thrown.

**Note:** If the <window-state> element is omitted, the portlet is assumed to support all standard WSRP window states. If wsrp:* is specified, it implies all of the standard wsrp: values. Support for "wsrp:normal" is always assumed, even if it is not explicitly indicated.

```
<supports>
   <mime-type>text/html</mime-type>
    <window-state><name>wsrp:normal</name></window-state>
    <window-state><name>wsrp:solo</name></window-state>
    <window-state><name>wsrp:maximized</name></window-state>
    <window-state><name>wsrp:minimized</name></window-state>

    <window-state>
      <name>urn-myproject-windowstates:header</name>
      <description lang="en">shows just the header for the
portlet</window-state>
    </window-state>

   ...
</supports>

<custom-items>
 <custom-window-state>
  <name>urn-myproject-windowstates:header</name>
  <description lang="en">shows just the header for the
portlet</description>
 </custom-window-state>
</custom-items>
```

2. Implement the window state in the portlet code.

   A portlet can access the window state via the `PortletInfo.WindowState` property during any portlet request, but may only change the current window state during a `performBlockingInteraction` (typically, an ASP.NET form postback). The new window state is returned in the response to the WSRP Consumer. If the WSRP Consumer does not understand the window state returned by a portlet, it can ignore it. If the WSRP Consumer requests a window state that the portlet does not understand, the WSRP Producer will map to wsrp:normal.

```
protected void returnButton_Click(object sender, EventArgs e)
{
 //set the window state back to wsrp:view
 WSRPPortletContext.Current.Portlet.WindowState = "wsrp:view";
```

```
}

protected void Page_PreRender(object sender, EventArgs e)
{
 PortletInfo portlet = WSRPPortletContext.Current.Portlet;
  if (portlet.WindowState == "wsrp:maximize" ||
portlet.WindowState == "wsrp:solo")
  {
   maximizeButton.Visible = false;
   returnButton.Visible = true;
  }
  else if (portlet.WindowState == "wsrp:view")
  {
   maximizeButton.Visible = true;
   returnButton.Visible = false;
  }
}
```

# Accessing WSRP Consumer Registration Information Using the .NET Portlet Toolkit

To identify a specific WSRP Consumer and access registration information, use the `RegistrationInfo` class in the .NET Portlet API.

By requiring registration, a WSRP Producer can request additional information about a WSRP Consumer which can be used to tailor the available portlets and resources. WSRP Consumer registration information is available to WSRP portlets through the `RegistrationInfo` class in the .NET Portlet API.

1. Configure the wsrp-producer.xml file to require registration, and define any registration properties. (The ConsumerAgent, ConsumerName and Handle are available by default.)

```
<?xml version="1.0"?>
<wsrp-producer
xmlns="http://www.bea.com/al/dotnet/wsrpproducer/1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" >

 <requires-registration>true</requires-registration>
```

```
 <registration-properties>
  <property name="ConsumerClass" type="xs:string">
   <label lang="en">Consumer Class</label>
   <hint lang="en" >the type of consumer</hint>
  </property>
 </registration-properties>
...
</wsrp-producer>
```

2. Access the registration information using the `RegistrationInfo` class in the .NET Portlet API. You can also use `RegistrationPropertyAttribute` to bind a registration property to a page member variable (field).as shown in the example below.

   The `RegistrationPropertyAttribute` preference attribute has two optional properties:

   - Key: A string that provides the key name of the property to which the member variable is bound. This value must match the value of the name attribute for the registration property defined in the wsrp-producer.xml configuration file. If this value is not defined, the key defaults to the name of the member variable on which it is defined. If the key value does not match a registration property defined in the wsrp-producer.xml file, the member variable will use the default value. Key values must be unique within a page.
   - DefaultValue: A string that provides the value to use when the preference is unavailable or has not yet been set. If this value is not provided, it will default to a value based upon the type of the variable as follows:

| Property Type | Default value if not assigned |
|---|---|
| string | String.Empty |
| int, long | 0 |
| double, float | 0.0 |
| bool | false |
| DateTime | DateTime.MinValue |

```
public partial class RegistrationPage : System.Web.UI.Page
{
    protected RegistrationInfo registration =
WSRPPortletContext.Current.ConsumerRegistration;

    [RegistrationProperty(Key = "doubleProperty", DefaultValue
 = 32)]
```

```
    protected double doubleProp;

    protected void Page_Load(object sender, EventArgs e)
    {
        registrationTable.Rows.Add(getPrefRow("ConsumerAgent",
 registration.ConsumerAgent));
        registrationTable.Rows.Add(getPrefRow("ConsumerName",
registration.ConsumerName));
        registrationTable.Rows.Add(getPrefRow("Handle",
registration.Handle));
        foreach (KeyValuePair<string, string> kvp in
registration.Properties)
            registrationTable.Rows.Add(getPrefRow("regprop:" +
 kvp.Key, kvp.Value));


registrationTable.Rows.Add(getPrefRow("RegistrationProperty:doubleProperty",
 doubleProp.ToString()));
    }

...
```

# Localizing WSRP Portlet Metadata

To provide localized portlet metadata, configure the WSRP Producer to use localized strings and provide the necessary resource files.

The Oracle WebCenter Application Accelerator for .NET supports standard ASP.NET mechanisms for localizing portlet markup. In addition, you can localize WSRP portlet metadata through the wsrp-producer.xml file. Localizable elements are those derived from the schema type “localizableStringType”. These are:

- portlet/description
- portlet/display-name
- portlet/portlet-info/title
- portlet/portlet-info/short-title
- property/label
- property/hint

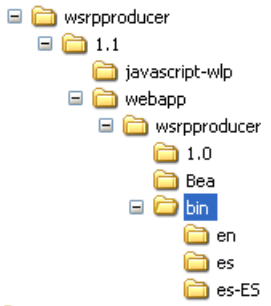
- portlet/supports/window-state/description
- portlet/supports/portlet-mode/description
- custom-portlet-mode/description
- custom-window-state/description

For a complete list of elements in the wsrp-producer.xml file, see *WSRP Producer Configuration Elements (wsrp-producer.xml)* on page 50.

1. Configure the portlet metadata in wsrp-producer.xml to use "key" attributes to reference strings. You can include a default language and value as shown in the example below..

```
<portlet>
...
 <display-name lang="en" key="my-display-name">My
Portlet</display-name>
 <description lang="en" key="my-description">Displays
data</description>
...
</portlet>
```

2. Create a subdirectory for each supported locale in the directory containing the WSRP Producer assembly. For example, if locales "es", "es-ES", and "en" are supported, you would use the following structure:




3. Create resource files that define localized strings for all supported languages and place them in the appropriate locale-specific directory. Resource files must be named according to the filename convention {resource-class}.{locale}.resources. for example myportlet.es.resources and wsrp-producer.es.resources.

   - For all resource strings that are declared as a descendent of a `<portlet>` element, the resource-class name is the value of the portlet handle.
   - For all other resource strings, the resource-class name is "wsrp-producer".

**Note:** .NET resource files must be created using the resgen.exe tool (included in Visual Studio and the .NET SDK).

# Logging .NET Application Accelerator Activity

To view logs of .NET Application Accelerator activity, use Oracle WebCenter Interaction Logging Spy.

Oracle WebCenter Interaction Logging Spy is automatically installed with Oracle WebCenter Interaction or installed on a remote server as part of the Oracle WebCenter Logging Utilities, a stand-alone package. Logging Spy listens for log messages transmitted using network broadcast. By configuring a list of message senders, you can listen in on log traffic from multiple Oracle WebCenter applications in a single console.

The .NET Application Accelerator includes four logging message senders:

| Logging Message Sender | Description |
|---|---|
| .NET WSRP Producer | The stand-alone web application designed to serve as a WSRP producer for Oracle WebLogic Portal. This name is not configurable and transmits all messages related to the WSRP Producer. The WSRP producer "loggingName" entry in Web.config is not currently used. |
| ALIPortletProject | Transmits log messages from the portlet ASP.NET web application. The name of this message sender is defined in the Web.config file of any ASP.NET portlet project built using the .NET Portlet Toolkit. The name of the sender defaults to the project name entered when the project was created, but can be changed by modifying the value attribute of the following entry:<br>`<add key="ptedk.LoggingApplicationName" value="ALIPortletProject"/>` |

| Logging Message Sender | Description |
|---|---|
| BEA Portlet API | Transmits log messages from the .NET Portlet API. |
| ALI .NET Application Accelerator | Reserved for future use. No logging traffic is sent by this sender. |

To add a .NET Application Accelerator message sender to your Spy configuration, follow the instructions below.

1. Open Logging Spy.
2. Select **View ➤ Set Filters** .
3. In the **Filter Settings** dialog, click **Edit ➤ Add Message Sender** and choose a message sender (see the table above).

For more information on configuring Logging Spy, see the *Oracle WebCenter Interaction Web Service Development Guide* and the *Administrator Guide for Oracle WebCenter Interaction*.

# Consuming a .NET WSRP Portlet in Oracle WebLogic Portal

To consume remote portlet resources in Oracle WebLogic Portal, you must create a Remote Portlet using either the IDE tools (described here) or Oracle WebLogic Portal online administration tools. Once created, Remote Portlets (.portlet files) can be added to a portal description (.portal file).

To create the Remote Portlet using the Oracle WebLogic Portal IDE tools, follow the steps below. For detailed instructions on using these tools, see the Oracle WebLogic Portal documentation.

**Note:** In order to produce portlets, the WSRP Producer must be installed and correctly configured within the local IIS instance. You must know the address of the WSRP Producer's WSRPService.wsdl file. The URL will vary depending on how IIS is configured, but it should be similar to the following: http://{wsrpproducer hostname:port}/wsrpproducer1.1/1.0/WSRPService.wsdl. To ensure that the WSRP Producer is running, paste this URL into the address bar of a web browser.

For detailed instructions on creating a WSRP portlet in Oracle WebLogic Portal, see *Creating Remote Portlets, Pages and Books* in the Oracle WebLogic documentation.

Oracle WebLogic Portal supports WSRP Preferences, but in order to use preferences, the .portal file describing a portal desktop in WebLogic Portal must be converted from "file" mode to "streaming" mode. For details, see *Creating a Desktop* in the Oracle WebLogic Portal documentation. Also, in order to modify preferences, a user must be logged into the portal. See the Oracle WebLogic Portal documentation for additional information on configuring login.

# Debugging WSRP Portlets

To debug WSRP portlets, use Oracle WebLogic Portal logging tools and check the list of common configuration problems.

If you encounter trouble accessing the WSRP Producer from the Oracle WebLogic Portal IDE or online administrative tools, attempt to access the WSRPService.wsdl using a web browser at this address: http://<IP-address>:<port-number>/wsrpproducer/1.0/WSRPService.wsdl If errors are returned, correct them; if no errors are returned, switch to the Oracle WebLogic Portal Eclipse tools and examine the error log.

Oracle WebLogic Portal also provides a SOAP monitor to track SOAP traffic between a WSRP Consumer and Producer. For detailed instructions on accessing the SOAP monitor, see *Using the Monitor Servlet* in the Oracle WebLogic Portal documentation. When using the SOAP monitor, it is helpful to create a new .portal file with a single portlet so you can view the traffic between the Consumer and a single Producer.

The following list provides solutions to common configuration problems.

- **Preferences don't work over WSRP in Oracle WebLogic Portal**: Ensure that the WebLogic .portal file has been converted to streaming mode. For details, see *Managing Portal Desktops* in the Oracle WebLogic Portal documentation.
- **Images do not appear in WSRP portlets**: The Oracle WebLogic Portal WSRP Consumer allows access to resources hosted in remote web applications acting as WSRP Producers. The Oracle WebCenter Application Accelerator for .NET WSRP Producer is a web application that is separate from the ASP.NET web sites being consumed. Oracle WebLogic Portal must be configured to allow URL-addressable content such as images, style sheets, and JavaScript includes to be returned to Oracle WebLogic Portal and users' web browsers. To do this, a `ResourceConnectionFilter` must be added to the portal applications as shown in the example below. This code sample is for Oracle WebLogic Portal 9.2.

```
<code>
package local;
import com.bea.wsrp.consumer.resource.ResourceConnectionFilter;
public final class AllowAllResourceConnectionFilter
 implements ResourceConnectionFilter
 {
```

```
  /**
     * Accept all resource URIs.
     * @return always returns <code>true</code>
     */
        public boolean allowedURL(String resourceURI) {
        return true;
        }
 }
</code>
```

This class is registered in Oracle WebLogic Portal by adding an entry in the appropriate location in the WEB-INF/web.xml file. For example, the following code registers the permissive `ResourceConnectionFilter` implemented above.

```
<!--  WLP 9.2 ResourceProxyServlet -->
<servlet>

<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-name>


<servlet-class>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-class>

  <init-param>
   <param-name>resourceConnectionFilter</param-name>

<param-value>local.AllowAllResourceConnectionFilter</param-value>

 </init-param>
</servlet>
 <servlet-mapping>

<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-name>

   <url-pattern>/resource/*</url-pattern >
 </servlet-mapping>
```

**Note:** This code is intended as an example and is not suitable for production. In production environments, we recommend that applications constrain allowable URLs to those that are known to produce remote ASP.NET web sites. For more information, see the Oracle WebLogic Portal 9.2 API documentation. For details on configuring a `ResourceConnectionFilter` in Oracle WebLogic Portal 8.1, see *Establishing WSRP Security* in the Oracle WebLogic Portal documentation.

- **Portlet styles and themes are not displayed**: Some portals strictly enforce the *HTML 4.01 DTD*. Portlets that use deprecated HTML might not be displayed correctly. Confirm the DTD used by the target portal and design your portlets accordingly.

#### Using SAML Token Authentication with Oracle WebLogic Portal

To implement single-sign on (SSO) authentication with Oracle WebLogic Portal using SAML tokens, you must configure the Oracle WebLogic Portal WSRP Consumer, the WSRP Producer, and the remote portlet application.

Oracle WebLogic Portal can be configured to send SAML assertions over WSRP. The SAML token is passed directly to the Remote Portlet Application as part of the HTTP headers. The SAML token can be accessed through HTTP Request Headers (`Request.Headers["SAMLToken"]`). The remote portlet host handles authentication using the Custom HttpModule "SAMLAuthenticationModule" and sets the user principal name for the request so that user gain access to remote portlets.

1. Add the SAML security policy declaration to the WSRPService.wsdl and wsrp_v1_bindings.wsdl files.
   a) Open the wsdl file for the WSRP Producer: \wsrpproducer\1.1\webapp\wsrpproducer\1.0\WSRPService.wsdl.
   b) If it is not already present, add the following policy declaration to the WSRPService.wsdl file as a child of the root element `<wsdl:definitions>` and before the `<wsdl:service>` element. (The WSRPService.wsdl file installed with the WSRP Producer includes the SAML policy by default.)

```
<wsp:Policy s1:Id="SAMLAuth.xml"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:s1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">

 <wssp:Identity>
  <wssp:SupportedTokens>
   <wssp:SecurityToken
TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SAMLAssertionID">

    <wssp:Claims>

<wssp:ConfirmationMethod>sender-vouches</wssp:ConfirmationMethod>

    </wssp:Claims>
   </wssp:SecurityToken>
  </wssp:SupportedTokens>
```

```
 </wssp:Identity>
</wsp:Policy>
```

c) Open the wsdl bindings file for the WSRP Producer: \wsrpproducer\1.1\webapp\wsrpproducer\1.0\wsrp_v1_bindings.wsdl.

d) Find the `<wsdl:input>` elements with the names "getMarkup" and "performBlockingInteraction". If not already present, add the `<Policy>` element shown below. (The bindings file installed with the WSRP Producer includes this code within comments; to enable the code, remove the comment tags and make sure the URI attribute matches the Id value of the SAML policy in the WSRPService.wsdl file..) The complete xml should look as follows:

```
<wsdl:input name="getMarkup">
 <soap:body use="literal"/>
  <wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <wsp:PolicyReference URI="#SAMLAuth.xml"/>
  </wsp:Policy>
</wsdl:input>

<wsdl:input name=" performBlockingInteraction ">
 <soap:body use="literal"/>
  <wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <wsp:PolicyReference URI="#SAMLAuth.xml"/>
  </wsp:Policy>
</wsdl:input>
```

**2.** Add a `<securityTokenManager>` element to the WSRP Producer's Web.config file as shown below.

```
<microsoft.web.services2>
 <security>
  <securityTokenManager
   type="Bea.BasicNoAuthSAMLTokenManager, WSRPService"
   xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
   qname="Assertion"
   />
 </security>
</microsoft.web.services2>
```

**3.** Generate a SAML credential certificate and configure the Oracle WebLogic Portal WSRP Consumer to use the generated key. For instructions, see *Establishing WSRP Security with SAML* and *Configuring Single-Sign-On with Web Browsers and HTTP Clients* in the Oracle

WebLogic Portal documentation. You must also configure the Relying Party properties as described below.

a) On the Management tab, click **Relying Parties**.
b) In the Relying Parties table, double click on **rp_00001**.
c) Ensure that Sign Assertions and Include Keyinfo are checked .
d) Click **Save**.

4. Configure the remote portlet application to verify the SAML token.
   a) Import the certificate used to generate the SAML token to **Local machine store – Enterprise Trusted certificates** on the machine that hosts the remote portlet application.
   b) Configure the Web.config file of the ASP.NET application to use the SAML authenticator.
      1. Under the `system.web` element, update the authentication node to use "None".
         ```
         <authentication mode="None" />
         ```
      2. Configure the SAMLAuthenitcationModule in the `httpModule` section of `system.web` as follows:
         ```
         <httpModules>
          <add name="SAMLAuthentication"
         type="BEA.Portlet.Authentication.SAMLAuthenticationModule,
          SAMLAuth"/>
         </httpModules>
         ```

#### Using UNT Authentication with Oracle WebLogic Portal

To implement single-sign on (SSO) with Oracle WebLogic Portal using UNT (User Name Token) authentication, you must configure the WSRP Producer.

UNT authentication can be used with IIS (Windows) authentication or ASP.NET Forms Authentication. The steps below cover both options.

Caution: Using UNT in the manner described below results in passwords being sent between the WSRP Consumer and the WSRP Producer in plain text. Ensure that the Consumer-Producer channel is secured by https before using this approach for transmitting a security token.

1. Enable authentication for your remote ASP.NET portlet. For more information on configuring ASP.NET authentication, consult MSDN and the following resources: *http://msdn2.microsoft.com/en-us/library/eeyk640h%28VS.80%29.aspx*, *http://msdn2.microsoft.com/en-us/library/ms978378.aspx*, *http://support.microsoft.com/kb/324274*.

2. Add the UNT security policy declaration to the WSRPService.wsdl and wsrp_v1_bindings.wsdl files.

   a) Open the wsdl file for the WSRP Producer: \wsrpproducer\1.1\webapp\wsrpproducer\1.0\WSRPService.wsdl.

   b) If it is not already present, add the following policy declaration to the WSRPService.wsdl file as a child of the root element (<wsdl:definitions>) and before the <wsdl:service> element. (The WSRPService.wsdl file installed with the WSRP Producer includes the UNT policy by default.)

```
<wsp:Policy s1:Id="UNTAuth.xml"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:s1="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">

 <wssp:Identity
xmlns:wssp="http://www.bea.com/wls90/security/policy">
  <wssp:SupportedTokens>
   <wssp:SecurityToken
TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken">

    <wssp:UsePassword
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText"/>

   </wssp:SecurityToken>
  </wssp:SupportedTokens>
 </wssp:Identity>
</wsp:Policy>
```

   c) Open the wsdl bindings file for the WSRP Producer: \wsrpproducer\1.1\webapp\wsrpproducer\1.0\wsrp_v1_bindings.wsdl.

   d) Find the <wsdl:input> elements with the names “getMarkup” and “performBlockingInteraction”. If not already present, add the <Policy> element shown below. (The bindings file installed with the WSRP Producer includes this code within comments; to enable the code, remove the comment tags.) The complete xml should look as follows:

```
<wsdl:input name="getMarkup">
 <soap:body use="literal"/>
  <wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <wsp:PolicyReference URI="#UNTAuth.xml"/>
  </wsp:Policy>
</wsdl:input>

<wsdl:input name=" performBlockingInteraction ">
```

```
 <soap:body use="literal"/>
  <wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <wsp:PolicyReference URI="#UNTAuth.xml"/>
  </wsp:Policy>
</wsdl:input>
```

3. Map each Oracle WebLogic Portal user to a user that can access the ASP.NET portlet application. For instructions, see *Configuring User Name Token Security: Configuring the Consumer* in the Oracle WebLogic Portal Federated Portals guide.
4. If you are using ASP.NET Forms Authentication, you must provide information about the login form to the WSRP Producer. In wsrp-producer.xml, add a `<forms-authentication>` entry as the last child element of the `<portlet>` element for each portlet that uses ASP.NET Forms Authentication. Each entry should include the following elements, as shown in the example that follows. For a full list of portlet configuration elements, see *WSRP Producer Configuration Elements (wsrp-producer.xml)* on page 50.

   **Note:** If you omit the `<forms-authentication>` element, the WSRP Producer will still try to authenticate to one of the IIS forms of authentication (Basic, Windows Integrated, or even Digest) if the ASP.NET is using one of those and the UNT is provided, but it will be unable to authenticate to ASP.NET Forms.

| Element | Description |
|---|---|
| `<IsSSOConfigured>` | Whether or not SSO through ASP.NET Forms Authentication is enabled or not (true or false). |
| `<loginpage>` | The name of the login page configured for ASP.NET Forms Authentication in the remote portlet application. |
| `<username-field-name>` | The name of the login form field that references the user name. |
| `<password-field-name>` | The name of the login form field that references the password. |
| `<login-button-name>` | The name of the submit button in the login form. |
| `<login-button-value>` | The value of the submit button in the login form. |
| `<always-use-anonymous-login>` | Whether or not to use anonymous login (true or false). If true, you must provide `<anonymous-username>` and `<anonymous-password>` elements. |

```
<portlet>
...
```

```
<forms-authentication>
 <IsSSOConfigured>true</IsSSOConfigured>
 <loginpage>login.aspx</loginpage>
 <username-field-name>txtUserName</username-field-name>
 <password-field-name>txtPassword</password-field-name>
 <login-button-name>cmdSubmit</login-button-name>
 <login-button-value>Submit</login-button-value>
 <always-use-anonymous-login>true</always-use-anonymous-login>

  <anonymous-username>userid</anonymous-username>
  <anonymous-password>pwd</anonymous-password>
</forms-authentication>
</portlet>
```

**Note:** The WSRP specification treats external resources used by a portlet application different from the portlet markup. An external resource is anything that is referenced by the portlet markup but not contained in the markup, such as externally referenced javascript files, images, or CSS style sheets. Portlet markup is retrieved from and proxied by the WSRP Producer; in the process the Producer can negotiate the authentication requirements of portlet applications. However, WSRP resources are retrieved directly by the WSRP Consumer without assistance from the WSRP Producer. As a result, external resources generally should not require authentication when using the WSRP Producer unless you can configure your WSRP Consumer to directly authenticate. If you are using Windows (IIS) authentication, you must move all external resources to a virtual directory or separate server that is configured to not require authentication in IIS. Take note that many ASP.NET controls and components use the WebResources feature to dynamically emit external javascript and image references in your markup. To make sure that WebResources references do not require authentication, ensure that any virtual paths from which these references originate do not require authentication.

# Debugging WSRP Portlets

To debug WSRP portlets, use Oracle WebLogic Portal logging tools and check the list of common configuration problems.

If you encounter trouble accessing the WSRP Producer from the Oracle WebLogic Portal IDE or online administrative tools, attempt to access the WSRPService.wsdl using a web browser at this address: http://<IP-address>:<port-number>/wsrpproducer/1.0/WSRPService.wsdl If errors are returned, correct them; if no errors are returned, switch to the Oracle WebLogic Portal Eclipse tools and examine the error log.

Oracle WebLogic Portal also provides a SOAP monitor to track SOAP traffic between a WSRP Consumer and Producer. For detailed instructions on accessing the SOAP monitor, see *Using the Monitor Servlet* in the Oracle WebLogic Portal documentation. When using the SOAP monitor, it is helpful to create a new .portal file with a single portlet so you can view the traffic between the Consumer and a single Producer.

The following list provides solutions to common configuration problems.

- **Preferences don't work over WSRP in Oracle WebLogic Portal**: Ensure that the WebLogic .portal file has been converted to streaming mode. For details, see *Managing Portal Desktops* in the Oracle WebLogic Portal documentation.
- **Images do not appear in WSRP portlets**: The Oracle WebLogic Portal WSRP Consumer allows access to resources hosted in remote web applications acting as WSRP Producers. The Oracle WebCenter Application Accelerator for .NET WSRP Producer is a web application that is separate from the ASP.NET web sites being consumed. Oracle WebLogic Portal must be configured to allow URL-addressable content such as images, style sheets, and JavaScript includes to be returned to Oracle WebLogic Portal and users' web browsers. To do this, a `ResourceConnectionFilter` must be added to the portal applications as shown in the example below. This code sample is for Oracle WebLogic Portal 9.2.

```
<code>
package local;
import com.bea.wsrp.consumer.resource.ResourceConnectionFilter;
public final class AllowAllResourceConnectionFilter
 implements ResourceConnectionFilter
 {
  /**
      * Accept all resource URIs.
      * @return always returns <code>true</code>
      */
         public boolean allowedURL(String resourceURI) {
         return true;
         }
 }
</code>
```

This class is registered in Oracle WebLogic Portal by adding an entry in the appropriate location in the WEB-INF/web.xml file. For example, the following code registers the permissive `ResourceConnectionFilter` implemented above.

```
<!--  WLP 9.2 ResourceProxyServlet -->
<servlet>

<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-name>
```

```
<servlet-class>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-class>

  <init-param>
   <param-name>resourceConnectionFilter</param-name>

<param-value>local.AllowAllResourceConnectionFilter</param-value>

 </init-param>
</servlet>
 <servlet-mapping>

<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-name>

   <url-pattern>/resource/*</url-pattern >
 </servlet-mapping>
```

**Note:** This code is intended as an example and is not suitable for production. In production environments, we recommend that applications constrain allowable URLs to those that are known to produce remote ASP.NET web sites. For more information, see the Oracle WebLogic Portal 9.2 API documentation. For details on configuring a `ResourceConnectionFilter` in Oracle WebLogic Portal 8.1, see *Establishing WSRP Security* in the Oracle WebLogic Portal documentation.

- **Portlet styles and themes are not displayed**: Some portals strictly enforce the *HTML 4.01 DTD*. Portlets that use deprecated HTML might not be displayed correctly. Confirm the DTD used by the target portal and design your portlets accordingly.

# Configuring WSRP Producer Service URLs (WSRPService.wsdl)

To update global WSRP Producer configuration settings after installation, update the WSRPService.wsdl file. Additional configuration of the WSRP Producer web site can be made via IIS administrative tools.

During installation, the WSRP Producer web site is deployed to IIS based on the configuration settings provided to the installer. This configuration information is used to parameterize the WSRP Producer's WSDL file, which describes the locations of various WSRP web services. After

installation has completed, any changes to the IP address, port number, or web site deployment path must be made to the WSRPService.wsdl file (located under \wsrpproducer\1.1\webapp\wsrpproducer\1.0\) If these settings are incorrect, the WSRP Producer will not be able to locate WSRP web services.

To update configuration settings, edit WSRPService.wsdl in a text editor and change the necessary entries. For example, to update the location of the WSRP Producer, change the following entry:

```
<soap:address
location="http://192.168.123.456:8888/wsrpproducer1.1/1.0/WSRPBaseService.asmx"/>
```

To configure portlets in the WSRP Producer, use the wsrp-producer.xml configuration file. For details, see *WSRP Producer Configuration Elements (wsrp-producer.xml)* on page 50.

# Registering a Portlet with the WSRP Producer

To deploy a portlet in the WSRP Producer, you must enter the URLs of the portlet content and additional metadata.

All the portlets available from the WSRP Producer must be registered in the wsrp-producer.xml file at the root of the WSRP Producer's web site. To add a portlet, edit the wsrp-producer.xml file in a text editor and add a `<portlet>` entry for each portlet within the `<portlets>` element. Once this step is complete, the portlet can be consumed by a WSRP Consumer. For a full list of elements in the wsrp-producer.xml file, see *WSRP Producer Configuration Elements (wsrp-producer.xml)* on page 50.

To upgrade an existing WSRP Producer configuration file (previously called portlets.xml), see Upgrading Existing WSRP Producer Implementations in the *Installation Guide for the Oracle WebCenter Application Accelerator for .NET*.

### WSRP Producer Configuration Elements (wsrp-producer.xml)

The wsrp-producer.xml file provides configuration information about all portlets deployed in the WSRP Producer. This file is also used to map portlet modes to specific URLs and define portlet info and portlet properties for use within portlets.

The table below summarizes the configuration elements required in wsrp-producer.xml. For a complete listing of elements and their restrictions, see the wsrp-producer.xsd schema file in the WSRP Producer installation directory (\wsrpproducer\1.1\webapp\wsrpproducer\).

**Note:** Some metadata elements can be localized using a “key” attribute. For details, see *Localizing WSRP Portlet Metadata* on page 36.

| Element | Description | Accepted Values |
|---|---|---|
| <requires-registration> | A flag indicating wether or not the WSRP Producer requires Consumers to register before providing a description of the portlets offered by the Producer. | true or false |
| <registration-property> | The list of registration properties required by the Producer. | For details on using registration properties, see *Accessing WSRP Consumer Registration Information Using the .NET Portlet Toolkit* on page 34. |
| <custom-items> | The list of custom window states and custom portlet modes that are used by the Producer. Custom items defined at this level may be referenced by any portlet offered by the Producer. Custom items that are shared by multiple portlets should always be defined here. | <custom-window-state> and <custom-portlet-mode><br>For details on window states and portlet modes, see *Implementing WSRP Portlet Window States Using the .NET Portlet Toolkit* on page 32 and *Implementing WSRP Portlet Modes Using the .NET Portlet Toolkit* on page 29. |
| <portlets> | A list of the portlets offered by the Producer. | <portlet> elements |
| <portlet> | Element within <portlets> that contains metadata for an individual portlet. | portlet configuration elements (see below) |
| **Portlet Configuration Elements** (used within each <portlet> element) | | |
| <handle> | WSRP portlet handle, a unique token used to refer to the portlet shared between the Consumer and Producer. | string |
| <url id=""> | Defines a URL for later use within portlet mode mappings; requires “id” parameter | URL |
| <expiration-cache> | Cache expiration setting in minutes | int |

| Element | Description | Accepted Values |
|---|---|---|
| <supported-locale> | A locale in which the portlet is able to generate markup. If not provided, indicates that the portlet will attempt to generate markup in any requested locale. | standard locale (as in Accept-Language header) |
| <description lang=" "> | A localizable description of the portlet intended for display in Consumer-generated dialogs. | string |
| <display-name lang=" "> | A localizable value intended for display in a Consumer's tooling for building aggregated pages. This value is usually shorter than either title element. | string |
| <supports> | Defines mime-type, window state, and portlet mode combinations supported by the portlet. If omitted, a default <supports> element for the mime-type"text/html" and all standard WSRP window-states and portlet modes is applied. | <mime-type>, <portlet-mode> and <window-state> |
| <mime-type> | Element within the <supports> element that defines the MIME type(s) supported by the portlet.<br><br>**Note:** If the <supports> element contains multiple <mime-type> elements, the specified portlet mode or window state settings apply to all. To specify different portlet modes or window states on a per MIME type basis, add multiple <supports> elements to the portlet definition, each with a different <mime-type> element. If the same <mime-type> is listed more than once, an exception is thrown. | In addition to fully-specified mime-types, '*' and type specific wildcards (e.g., 'text/*') may be used. |

| Element | Description | Accepted Values |
|---|---|---|
| <portlet-mode><br><name></name><br><url></url> | Defines support for a portlet mode for the mime-type(s) of the parent <supports> element. If omitted, defaults to supporting all standard wsrp portlet modes. | The <name> element must contain a standard WSRP portlet mode (wsrp:view, wsrp:edit, wsrp:help or wsrp:preview) or a URI to a custom portlet mode. Custom portlet modes should include a <description>.<br><br>Each supported mode must have an associated URL. Either include the URL for the mode within the <url> node, or use the idref attribute to refer to a URL defined as a child of the <portlet> element (see above). If neither an URL nor the idref is specified, the URL is assumed to be the first one found for the portlet.<br><br>For details on using portlet modes, see *Implementing WSRP Portlet Modes Using the .NET Portlet Toolkit* on page 29. |
| <window-state><br><name></name> | Defines support for a window state for the mime-type(s) of the parent <supports> element. If omitted, defaults to supporting all standard wsrp window states. | The <name> element must contain a standard WSRP window state (wsrp:normal, wsrp:solo, wsrp:minimized or wsrp:maximized) or a URI to a custom window state. Custom window states should include a description.<br><br>For details on using window states, see *Implementing WSRP Portlet Window States Using the .NET Portlet Toolkit* on page 32. |
| <portlet-info> | Localized portlet metadata intended for use in Consumer-generated dialogs. | <title>: Localized title for the portlet. This value is intended for display in a titlebar decoration for the portlet markup. |

| Element | Description | Accepted Values |
|---|---|---|
| | | <short-title>: Localized short title for the portlet.<br><br>For details on localization, see *Localizing WSRP Portlet Metadata* on page 36. |
| <portlet-property> | A list of portlet properties supported by instances of the portlet. | Each property element contains the following attributes: name: A unique name for the property used for indexing the property in a collection. (Required.)label: A short, human-readable name for the property intended for display in any Consumer-generated user interface for administering the portlet.hint: A short description of the property intended for display.type: The datatype of the property. Defaults to xs:string if omitted.defaultValue: The value sent to the portlet for this property if the property has not been explicitly set for the portlet instance.<br><br>For details on using portlet properties, see *Manipulating WSRP Properties Using the .NET Portlet ToolkitTo use WSRP properties in a portlet, configure the properties in the WSRP Producer and use the .NET Portlet API. To access and define properties, you can use a property attribute or a property collection.* . |
| <user-profile> | A list of user profile properties requested by the portlet. | <item>: The full name of the user profile property. |

| Element | Description | Accepted Values |
|---|---|---|
| <formsauthentication> | Defines support and configuration for SSO forms authentication. | For details on using this element, see *Using UNT Authentication with Oracle WebLogic Portal* on page 44. |

The example below defines two portlets. The first uses one URL for all portlet modes, while the second uses separate URLs for the view mode and edit mode. This code is from the WSRP sample code provided with the Oracle WebCenter Application Accelerator for .NET.

```
<?xml version="1.0"?>
<wsrp-producer
xmlns="http://www.bea.com/al/dotnet/wsrpproducer/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <requires-registration>false</requires-registration>

 <portlets>

  <portlet>
   <handle>quote</handle>
   <url
id="default">http://localhost:4614/WSRPSamples/StockQuotePortlet.aspx</url>

   <expiration-cache>10</expiration-cache>
   <supported-locale>en</supported-locale>
   <description lang="en">Portlet that displays stock
quotes.</description>
   <display-name lang="en">Stock Quote Portlet</display-name>
   <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>
     <name>wsrp:view</name>
     <url idref="default"/>
    </portlet-mode>
    <portlet-mode>
     <name>wsrp:edit</name>
     <url idref="default"/>
    </portlet-mode>
    <portlet-mode>
     <name>wsrp:help</name>
     <url idref="default"/>
    </portlet-mode>
```

```
    </supports>
    <portlet-info>
     <title lang="en">Stock Quote Portlet</title>
     <short-title lang="en">quote</short-title>
    </portlet-info>
    <portlet-properties>
     <property name="stockSymbols" type="xs:string"
defaultValue="BEAS"/>
    </portlet-properties>
   </portlet>

   <portlet>
    <handle>quote2</handle>
    <url
id="view">http://localhost:4614/WSRPSamples/StockQuotePortlet2/View.aspx</url>

    <url
id="edit">http://localhost:4614/WSRPSamples/StockQuotePortlet2/Edit.aspx</url>

    <expiration-cache>10</expiration-cache>
    <supported-locale>en</supported-locale>
   <description lang="en">Portlet that displays stock quotes using
 a seperate edit page.</description>
    <display-name lang="en">Stock Quote Portlet 2</display-name>
    <supports>
     <mime-type>text/html</mime-type>
     <portlet-mode>
      <name>wsrp:view</name>
      <url idref="view"/>
     </portlet-mode>
     <portlet-mode>
      <name>wsrp:edit</name>
      <url idref="edit"/>
     </portlet-mode>
    </supports>
    <portlet-info>
     <title lang="en">Stock Quote Portlet 2</title>
     <short-title lang="en">quote#2</short-title>
    </portlet-info>
    <portlet-properties>
     <property name="stockSymbols" type="xs:string"
defaultValue="BEAS"/>
    </portlet-properties>
   </portlet>
```

```
 </portlets>
</wsrp-producer>
```