Oracle® Fusion Middleware

Development Guide for Oracle WebCenter WSRP Producer for .NET

11*g* Release 1 (11.1.1.5.0)

E14117-04

April 2011

Oracle Fusion Middleware Development Guide for Oracle WebCenter WSRP Producer for .NET, 11*g* Release 1 (11.1.1.5.0)

E14117-04

Primary Author: Jennifer Shipman

Contributing Author: Promila Chitkara

Contents

Preface

Development Guide for Oracle WebCenter WSRP Producer for .NET provides detailed information on authoring portlets using the Oracle WebCenter WSRP Producer for .NET.

Audience

This document is intended for Oracle WebCenter developers who want to create WSRP portlets using Oracle WebCenter WSRP Producer for .NET.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/us/corporate/accessibility/index.html`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle WebCenter Application Accelerator for Microsoft .NET*
- Release notes

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introducing Oracle WebCenter WSRP Producer for .NET

This chapter introduces you to Oracle WebCenter WSRP Producer for .NET.

1.1 About the Oracle WebCenter WSRP Producer for .NET

The Oracle WebCenter WSRP Producer for .NET is a collection of libraries that surface ASP.NET applications and WebParts as WSRP 1.0 or 2.0 portlets. This release supports the latest ASP.NET 3.5 features such as Ajax and allows developers to code in ASP.NET without coding against a .NET WSRP framework.

1.2 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 WSRP Producer for .NET.
Release Notes	The release notes provide information about new features, issues addressed, and known issues in the release.

2 Understanding the Configuration File

This chapter describes contents of the wsrp-producer.xml file.

The set up of a Web Part or ASP.NET application as a WSRP portlet is not complete until the configuration file, wsrp-producer.xml, is filled out properly. Understanding this file is critical. Open up the wsrp-producer.xml installed in `<installdir>\wsrpdefault` for reference.

- Section 2.1, "Portlet Nodes"
- Section 2.2, "Portlet Content Transformation Rulesets"

2.1 Portlet Nodes

Each Web Part or .NET web application you wish to expose as a portlet must have its own `<portlet>` node. Portlet nodes can include the following elements:

- **Portlet handle:** The `<handle>` must be a string that uniquely identifies the portlet. Alphanumeric characters are suggested for this portlet property.
- **Supported locale:** The `<supported-locale>` must be a comma-delimited list of supported locales. The typical format is `[2 char language code] "-" [2 char country code]` (for example, `en-US`). This information is sent to the WSRP Consumer portal during registration. These values also drive the localization for core properties of the portlet such as title and description.
- **Localized portlet information:** The `<description>`, `<display-name>`, `<title>`, and `<short-title>` properties are localized according to the "lang" attribute value specified. Each value is returned as a LocalizedString for the corresponding WSRP values of description, displayName, title, and shortTitle. See the PortletDescription type in the WSRP specification for more details on these properties.
- **Supported modes:** The `<supports>` element includes several sub-elements:
  - **Portlet modes:** The `<portlet-modes>` element defines URLs (typically a relative path) for relevant WSRP portlet modes. The Oracle WebCenter WSRP Producer for .NET handles portlet modes as follows:

Table 2–1 Portlet Modes

Portlet mode	Action
wsrp:view	Render the portlet.
wsrp:edit	Render the portlet editor.
wsrp:help	Render a web page for help.

Table 2–1 (Cont.) Portlet Modes

Portlet mode	Action
wsrp:preview	This mode is treated the same as wsrp:view.
urn:javax:portlet:mode:custom:edit_defaults	This WSRP custom mode is for WebCenter customization.

Any mode not on this list will be treated like wsrp:view mode. See the sections that follow for examples of each portlet mode.

– **MIME type:** The `<mime-type>` element defines a mime type supported by the portlet. If all mime types are supported, pass in "*". See the WSRP Specification under the MarkupType type for more information on this setting.

- **User profile information:** The `<user-profile>` element allows you to define user profile information in `<profile-item>` elements for use in portlets. For more information on using this element, see Section 5.8, "Adding User Profile Data."
- **Portlet content transformation:** The `<RuleSetRef>` section allows you to associate existing transformation rules (defined outside the portlet node) with a portlet. Use the ID defined in the associated `<RuleSet>` element. Multiple rulesets can be referenced by placing the IDs in the appropriate order in a comma-delimited string. To add your own transformation rules, add a `<RewriterConfig>` element and follow the same structure defined for the `<Rules>` element. Existing rules can be combined with custom rules as shown in the example below:

```
<RuleSetRef>Default</RuleSetRef>
<RewriterConfig>
  <Rules>
    <RewriterRule>
      <LookFor>test.csv</LookFor>
      <ChangeTo>download.csv</ChangeTo>
      <ChangeToAbsolute>false</ChangeToAbsolute>
      <ApplyTo>*</ApplyTo>
      <MakeResource>false</MakeResource>
    </RewriterRule>
  </Rules>
</RewriterConfig >
```

Note: If custom rules are combined with existing rulesets, custom rules will be applied in order of listing after all the referenced rulesets.

In the above example, the portlet will apply all rules associated with the "Default" rule set and then replace any occurrences of "test.csv" with "download.csv".

For details on rulesets, see the next section.

2.2 Portlet Content Transformation Rulesets

The Oracle WebCenter WSRP Producer for .NET utilizes regular expressions to transform relative URLs to absolute URLs and to URLs proxied through the WSRP Consumer if necessary. Specific transformation rules are defined in the wsrp-producer.xml file in `<RuleSet>` elements and referenced in each portlet. Rulesets are a way of clustering a group of markup transformation rules.

By default, there is a single default ruleset that ensures that hrefs are transformed and proxied as WSRP resources, but "src" URL references (images and javascript) are transformed but not proxied. If you want to use different transformation settings, simply create a new ruleset as described in the next section. For an example of using rulesets in a portlet, see Section 5.5, "Using Markup Transformation."

Note: It is strongly recommended that you not modify the default ruleset.

2.2.1 Creating Custom Rules

As noted in the previous section, custom rules can be defined within the portlet node or outside the portlet node in a `<RuleSet>` element. Rulesets can be very useful if you have a set of portlets that should use the same rules. Instead of copying the URL transformation rules for each portlet, you can create a single ruleset and reference the ID in each portlet. The syntax rules for the `<RewriterConfig>` element are the same for both locations. (If the rule is defined within an individual portlet node, the `<RuleSet>` element is omitted.)

The `<RewriterConfig>` element contains a `<Rules>` element that contains `<RewriterRule>` elements, which define transformation rules using the elements in the following table.

Table 2–2 Elements to Define Transformation Rules

Element	Value/Description
`<LookFor>`	Required. Defines the URL to search for using text and/or regular expressions. For example, `<LookFor>(activeSrc\":\")([^\"]+)</LookFor>` indicates that the transformer should look for URLs like "activeSrc":"./test/foo/bar.js" or "activeSrc":"test2/bas.html" for replacement. `<LookFor>`must be the first element in the `<RewriterRule>` element.
`<ChangeToAbsolute>`	Required. If set to **true**, transforms any URL identified with the `<LookFor>` element into an absolute URL. (Typically, the transformation you wish to perform on an URL is to make it an absolute URL since a relative URL does not make sense when proxied through a WSRP Consumer.)
`<MakeResource>`	Required. If set to **true**, ensures that the URL is proxied by the WSRP Consumer. While this is important for resources that you wish to protect, applying this option liberally can have significant performance ramifications since the WSRP Consumer must rewrite the URL and browser caching cannot be relied upon. Use the `<ApplyTo>` to ensure that only those URLs you wish to proxy are included.
`<ApplyTo>`	Limits the URLs that are transformed to WSRP resources and proxied using a comma-delimited list of keywords. (All URLs returned by the `<LookFor>` element will be transformed to absolute URLs if the `</ChangeToAbsolute>` is included in the rule.)

Table 2–2 (Cont.) Elements to Define Transformation Rules

Element	Value/Description

`<ChangeTo>`	Changes the text returned by the `<LookFor>` element to the provided text. (This is not used for URL transformation, but allows you to use the `<RewriterRule>` element to change any text within the portlet, and can include regular expressions.) For example, the entry below redirects traffic from myhost to myhost.mycompany.com instead:
	`<RewriterRule>`
	`  <LookFor>myhost[\:]*[0-9]* </LookFor>`
	`  <ChangeTo>myhost.mycompany.com</ChangeTo>`
	`</RewriterRule>`

3 Managing Required Script Files

This chapter discusses `jquery-1.3.2.min.js` and `msPortletResizeHelper.js` files.

3.1 Managing Required Script Files

The WSRP Producer for .NET makes use of two files: `jquery-1.3.2.min.js` and `msPortletResizeHelper.js`. By default, these files are made into WSRP resources. Unlike any other javascript files portlet content transformation rulesets described in Section 2.2, "Portlet Content Transformation Rulesets" are not applied to these files. To take advantage of browser caching of javascript you may wish to deploy these required scripts to a web server, such as Apache.

To override the default behavior, add the `jQueryExternalURL` and `msResizeHelperExternalURL` .NET application settings keys, as shown in Example 3–1, to the `web.config` file of your WSRP Producer for .NET.

Example 3–1 .NET Application Settings Keys

```
<appSettings>
      <add key="jQueryExternalURL"
value="http://myhost.mycompany.com/path/jquery-1.3.2.min.js"/>
     <add key="msResizeHelperExternalURL"
value="http://myhost.mycompany.com/path/msPortletResizeHelper.js"/>
</appSettings>
```

4 Configuring Security Options

The WSRP specification does not specify which security techniques should be employed, but does encourage anyone using message-level security to use WS-Security standards. The specification makes mention of transport-level security, but only requires that "a Producer's WSDL declare ports for an HTTPS service entry point".

- Section 4.1, "WS-Security (WSS) Configuration"
- Section 4.2, "SSL Configuration"

4.1 WS-Security (WSS) Configuration

The Oracle WebCenter WSRP Producer for .NET supports two types of WS-Security token profiles: SAML and Username Token (UNT).

4.1.1 WSDL Configuration

There are two WSDL files that contain WSS policy information:

- **WSRP 1.0**: `wsrp_v1_bindings.wsdl` and `WSRPService.wsdl` (`<installdir>\wsrpdefault\wsdl\1.0`)
- **WSRP 2.0**: `wsrp-2.0-bindings.wsdl` and `WSRPService.wsdl` (`<installdir>\wsrpdefault\wsdl\2.0`)

The WSRPService.wsdl file should include the following as a child of the root element `<wsdl:definitions>` and before the `<wsdl:service>` element:

Example 4–1 SAML

```
<wsp:UsingPolicy wsdl:Required="true" />
<wsp:Policy s1:Id="SAMLAuth.xml">
    <wsp:All>
      <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:All>
</wsp:Policy>
```

Example 4–2 UNT

```
<wsp:UsingPolicy wsdl:Required="true" />
  <wsp:Policy s1:Id="UNTAuth.xml">
    <wsp:All>
      <wssp:Identity>
        <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-profi
le-1.0#PasswordText"/>
          </wssp:SecurityToken>
        </wssp:SupportedTokens>
      </wssp:Identity>
    </wsp:All>
  </wsp:Policy>
```

The UsingPolicy must be set to "required" or certain WSRP Consumers (such as WLP) will not honor the security policy. The Policy node is a standard WS-Security policy element where SAML v1.1 sender-vouches confirmation is defined. See the WS-Security specification for details.

The most important piece to note for WSDL configuration is the "Id" attribute for the policy. This must be referenced in the wsrp*bindings.wsdl file. Open the wsrp_v1_bindings.wsdl or wsrp-2.0-bindings.wsdl file and find the `<wsdl:input>` elements with the names "getMarkup" and "performBlockingInteraction". These sections should reference which policy to use. For example, if the Id for the policy in WSRPService is "SAMLAuth.xml", the policy reference in wsrp*bindings.wsdl would look like the following:

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
 <wsp:PolicyReference URI="#SAMLAuth.xml"/>
</wsp:Policy>
```

(If the Id for the policy in WSRPService.wsdl is "UNTAuth.xml", the PolicyReference URI value in wsrp*bindings.wsdl would be "#UNTAuth.xml".)

4.1.2 Web.config Configuration

The Oracle WebCenter WSRP Producer for .NET uses Microsoft Web Services Extensions (WSE) 2.0 SP3. This section discusses the default way in which WSE is set up to handle WS-Security tokens. To learn more about WSE and security tokens, refer to the Microsoft documentation (`http://msdn.microsoft.com/en-us/library/ms824677.aspx`).

Security aspects of WSE are managed under the `<security>` element under the `<microsoft.web.services2>` element in Web.config.

The default UNT and SAML Security Token Managers defined in Web.config do not actually perform authentication. Instead, they strip off the token XML and provide it to individual portlets registered under the Oracle WebCenter WSRP Producer for .NET WSRP producer. This allows individual portlets to implement their own authentication and authorization methodologies. To implement authentication at the WSRP SOAP endpoints, create your own SecurityTokenManager for WSE following the Microsoft documentation.

Note: WSE does not properly handle WSS 1.0 security tokens sent by the WebCenter WSRP Consumer, so the Oracle WebCenter WSRP Producer for .NET uses a request filter to decrypt any encrypted content, validate signatures, and then pass the modified SOAP message without encryption and signature to WSE. To learn more about the default request filter implemented for security tokens sent from WebCenter, see the development tutorials in the next section.

To learn more about the default UNT and Security Token Managers, see Section 5.9, "Setting Up Security Tokens" in the next section. Source code for the default SecurityTokenManager is available in the `\wsrpdefault\src` folder in the installation directory.

4.2 SSL Configuration

To set up SSL (either one-way or two-way), follow the Microsoft documentation on configuring IIS and the documentation provided for your WSRP Consumer portal. Use "https" on all endpoint URLs in WSRPService.wsdl. For WSRP resources, the Oracle WebCenter WSRP Producer for .NET will determine whether to use http or https depending on which was used to access the endpoint.

Note: Two-way SSL is not supported in Oracle WebCenter.

---

5 Accomplishing Common Development Scenarios

This chapter provides step-by-step tutorials to achieve common development scenarios:

- Section 5.1, "Adding a Web Part as a Portlet"
- Section 5.2, "Adding a Web Part with ASP.NET AJAX Controls"
- Section 5.3, "Adding an ASP.NET Application"
- Section 5.4, "Adding a Help Page"
- Section 5.5, "Using Markup Transformation"
- Section 5.6, "Using Portlet Styles"
- Section 5.7, "Using Interportlet Communication with JavaScript"
- Section 5.8, "Adding User Profile Data"
- Section 5.9, "Setting Up Security Tokens"
- Section 5.10, "Using Oracle WebCenter Features"

5.1 Adding a Web Part as a Portlet

By default, the ASP.NET Web Part editor zone supports three different types of preferences (referred to as Personalization). This section illustrates how to work with the default preferences. The Oracle WebCenter WSRP Producer for .NET also supports custom Personalization editors.

In the example below, steps 1-3 create an ASP.NET Web Part just as you would for consumption in ASP.NET outside of the Oracle WebCenter WSRP Producer for .NET, and steps 5-8 add the Web Part to the Oracle WebCenter WSRP Producer for .NET. Refer to the Microsoft documentation for more details on how to create Web Parts from User Controls (`http://msdn.microsoft.com/en-us/magazine/cc300767.aspx`).

1. Open a Visual Studio Web Application Project and create a Web User Control named TestPreferences.ascx.
2. Add three labels named lblStringPref, lblBooleanPref, and lblEnumPref.
3. Create three properties that include the `[Personalizable, WebBrowsable]` attribute. Match the label display values to the properties as shown in the example code below:

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    lblBooleanPref.Text = "Checkbox Pref: " + _boolPref;
    lblStringPref.Text = "Text Pref: " + _stringPref;
    lblEnumPref.Text = "Dropdown Pref: " + _enumPref;

}
#region Preferences
private string _stringPref = "";
private bool _boolPref = false;

public enum TestEnum
{
    ListItem1, ListItem2, ListItem3
}
private TestEnum _enumPref = TestEnum.ListItem2;

[Personalizable, WebBrowsable]
public string TextPref
{
    get { return _stringPref; }
    set { _stringPref = value; }
}

[Personalizable, WebBrowsable]
public bool CheckboxPref
{
    get { return _boolPref; }
    set { _boolPref = value; }
}

[Personalizable, WebBrowsable]
public TestEnum DropdownPrefs
{
    get { return _enumPref; }
    set { _enumPref = value; }
}
#endregion
```

4. Test that the code compiles before continuing.
5. Make a copy of the existing \ASPNET_AJAX_sample folder under <installdir>\wsrpdefault\portlets and rename it "TestPreferences".
6. Replace the Simple_DateTime.ascx file with the TestPreferences.ascx file you created in steps 1-3.
7. Copy the compiled assembly from your Visual Studio project to the <installdir>\wsrpdefault\bin folder.
8. Update the default.aspx file in <installdir>\wsrpdefault\portlets\TestPreferences to reference your control instead of the Simple_Date_Time control. The resulting code should look something like the following:

```
<%@ Page Language="C#"
MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
Page Test" %>
<%@ Register src= "~/portlets/TestPreferences/UserControl/TestPreferences.ascx"
tagname="TestPreferences" tagprefix="uc1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="webpartcontrol"
Runat="Server">
    <uc1:TestPreferences ID="TestPreferences1" runat="server" />
</asp:Content>
```

9. Update the <installdir>\wsrpdefault\wsrp-producer.xml file to include your portlet as follows:

   - Make a copy of an existing `<portlet>` element and its contents.
   - Change the handle, display name, description, title, and short title elements to the correct values for the new portlet.
   - Update the `<url>` node under the `<portlet-mode>` element to point to your portlet.

   For the example above, the`<portlet>` element would look like the following:

```
<portlet>
 <handle>TestPreferences</handle>
 <supported-locale>en,fr</supported-locale>
 <descriptions>
  <description lang="en">Test Preferences</description>
  <description lang="fr">Test Preferences</description>
 </descriptions>
 <display-names>
  <display-name lang="en">Test Preferences</display-name>
  <display-name lang="fr">Test Preferences</display-name>
 </display-names>
 <titles>
  <title lang="en">Test Preferences</title>
  <title lang="fr">Test Preferences</title>
 </titles>
 <short-titles>
  <short-title lang="en">Test Preferences</short-title>
  <short-title lang="fr">Test Preferences</short-title>
 </short-titles>
 <supports>
  <portlet-modes>
   <portlet-mode>
    <name>wsrp:view</name>
    <url>/portlets/TestPreferences/default.aspx</url>
   </portlet-mode>
   <portlet-mode>
    <name>wsrp:edit</name>
    <url>/portlets/TestPreferences/default.aspx</url>
   </portlet-mode>
  </portlet-modes>
  <mime-type>text/html</mime-type>
 </supports>
 <user-profile>
  <profile-item>businessInfo/online/email</profile-item>
 </user-profile>
</portlet>
```

10. Register the WSRP producer by pointing your WSRP Consumer at the WSDL URL that is in the shortcut Start | Programs | Oracle Application Accelerator for .NET | WSRP 1.0 WSDL. (Refer to the product documentation for your WSRP Consumer portal for details on how to register a WSRP producer WSDL.)

11. The portlet should now be available in your WSRP Consumer. Test the portlet functionality:

   a. Add the portlet to your portal. You should see the three labels showing the default values set for the preferences in the TestPreferences.ascx file.

   b. Access the preference editor according to the documentation for your WSRP Consumer portal. You should see the same preference editor you would see if you added your own Web Part editor zone in ASP.NET. Change the preferences, click Close, and observe that the labels change.

c. Log out and log in to your portal. The preference values you changed should be persisted.

5.2 Adding a Web Part with ASP.NET AJAX Controls

A key feature of the Oracle WebCenter WSRP Producer for .NET is support for controls that are rendered using ASP.NET AJAX.

This tutorial starts with the example in the previous section. Again, most of the steps are typical tasks required to set up an ASP.NET Web Part to use ASP.NET AJAX functionality. Refer to the Microsoft documentation for details on how to set up ASP.NET AJAX script managers and update panels.

1. Open the project you created in Section 5.1, "Adding a Web Part as a Portlet".
2. From the AJAX Extensions portion of the Visual Studio toolbox, add a Script Manager and an Update Panel.
3. Add a label, a text box, and a button inside the Update Panel.
4. Add a label, a text box, and a button outside the Update Panel. When you are finished, the source view should look something like the following:

```
<asp:Label ID="lblStringPref" runat="server" Text="Label"></asp:Label><br />
<asp:Label ID="lblBooleanPref" runat="server" Text="Label"></asp:Label><br />
<asp:Label ID="lblEnumPref" runat="server" Text="Label"></asp:Label><br />
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
    <asp:TextBox ID="txtAjax" runat="server"></asp:TextBox>
    <asp:Button ID="btnAjax" runat="server" Text="PostAJAX" /><br />
    <asp:Label ID="lblAjax" runat="server" Text="AJAX post here"></asp:Label>
</ContentTemplate>
</asp:UpdatePanel>
<asp:TextBox ID="txtPostback" runat="server"></asp:TextBox>
<asp:Button ID="btnPostback" runat="server" Text="Postback" /><br />
<asp:Label ID="lblPostback" runat="server" Text="Postback content
here"></asp:Label>
```

5. Modify the page load method so that text box entries are displayed in the label. The source code for the page load method will look something like the following:

```
protected void Page_Load(object sender, EventArgs e)
        {
            lblBooleanPref.Text = "Checkbox Pref: " + _boolPref;
            lblStringPref.Text = "Text Pref: " + _stringPref;
            lblEnumPref.Text = "Dropdown Pref: " + _enumPref;

            lblAjax.Text = txtAjax.Text;
            lblPostback.Text = txtPostback.Text;

  }
```

6. Test that the code compiles before continuing. (You can also register the control in an aspx page in your Visual Studio project to test that it is working. See the Visual Studio documentation for details.)
7. Replace the <installdir>\wsrpdefault\portlets\TestPreferences.ascx file with the latest version.

8. Open the TestPreferences.ascx file in a text editor and remove the `asp:ScriptManager` control. The Oracle WebCenter WSRP Producer for .NET references an ASP.NET Master Page that registers a ScriptManager, and ASP.NET does not allow more than one ScriptManager per application. See the Microsoft documentation on Master Pages and ScriptManager for more information.
9. Replace the compiled assembly in the <installdir>\wsrpdefault\bin folder.
10. Go back to your portal, refresh the portlet and test its functionality again.

5.2.1 Managing AJAX Outside the .NET Framework

There may be times when you want to control XMLHttpRequests directly, rather than rely on the ASP.NET 3.5 AJAX controls. To do this, you must account for the fact that URLs are transformed via the WSRP proxy. This tutorial provides the steps necessary to get a simple AJAX project working.

This example starts with the code from the previous example. Under the `asp:Label` in the previous sample, add the following code:

```
<!-- Added for simple AJAX sample -->
<br />
<asp:ListBox ID="listBoxPets" runat="server">
    <asp:ListItem Text="cat" Value="cat" Selected="True"/>
    <asp:ListItem Text="dog" Value="dog" />
    <asp:ListItem Text="bird" Value="bird" />
    <asp:ListItem Text="fish" Value="fish" />
</asp:ListBox>
<!-- This href URL will be properly transformed by WSRP proxy -->
<a id="hiddenposturl" href="AjaxPostDestination.aspx" visible="false" ></a>
<br />
<!-- Result of selection will be placed here -->
<div id="AjaxResponse"></div>
```

This code constructs a simple list box with a hidden href that points at the URL to which the XMLHttpRequest will send data. This URL is placed in an anchor href so that the markup transformer will rewrite the URL for the WSRP proxy. (For details on transformation, see Section 5.5, "Using Markup Transformation".)

Note: In this sample, there are no URLs returned in the response. However, you may need the response from the AJAX request to be transformed by the Portlet Content Transformation rulesets (for example, if you have URLs you want proxied by the WSRP Consumer). To do this, add a "handle" query string argument to the target URL that references the handle you gave the portlet in wsrp-producer.xml. In this sample, the handle would be "TestPreferences" so the hidden URL would be changed to `<a id="hiddenposturl" href="AjaxPostDestination.aspx?handle=TestPreferences" visible="false" ></a>`. You must also make sure that the Content Type returned is text/html (it is text/plain in this sample) or certain WSRP consumers will not properly proxy the transformed URLs.

Under the `<div>` add the following javascript:

```
<script language="javascript" type="text/javascript">
    var xmlHttp;
    /*  Tap into the onchange event so that whenever an item is selected in the
    list box, that value is sent to the server.
    */
    var selectmenu;
    var selectMenus = getElementsByTagNameAndEndingId("select", "listBoxPets");
    selectmenu = selectMenus[0];

    selectmenu.onchange = function() {
        // Prepare the XHR
        createXMLHttpRequest();
        // Get the selected value
        var selectedPet = getSelectedPet();
        // get the URL from a hidden argument since this will be transformed.
        var hiddenPetURL = getElementsByTagNameAndEndingId("a", "hiddenposturl");
        // Prepare the request.
        xmlHttp.open("POST", hiddenPetURL, true);
        // Assign this to a function that will process the requested items
        xmlHttp.onreadystatechange = handleStateChange;
        // Important note - If you use application/x-www-form-urlencoded you must
include the ;charset=UTF-8 or
        // this will not work in Firefox 3.x.  Works fine in all versions of IE
w/o this.
        xmlHttp.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded;charset=UTF-8");
        // Send the value as post parameters.  This is the suggested methodology
using the Application Accelerator for .NET
        xmlHttp.send("petselection=" + selectedPet);
    }

    /*
    Standard AJAX method.  This creates the appropriate XMLHttpRequest (XHR)
object, the core of AJAX.
    */
    function createXMLHttpRequest() {
        if (window.ActiveXObject) {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        else {
            xmlHttp = new XMLHttpRequest();
        }
    }

    /*
    This function handles the results of the XMLHttpRequest (XHR).
    This is part of the standard AJAX methodology of using the onreadystatechange
event of the XHR to manage the results.
    */
    function handleStateChange() {
        if (xmlHttp.readyState == 4) {
            if (xmlHttp.status == 200) {
                parseResults();
            }
        }
    }

    /*
    This helper function grabs the first selection and returns it
    */
    function getSelectedPet() {
        var result = "";
        var options = selectmenu.childNodes;
        var option = null;
        for (var i = 0; i < options.length; i++) {
            option = options[i];
            if (option.selected) {
                result = option.value;
            }
        }
        return result;
    }

    /*
    Write out the results returned from the XHR POST
    */
    function parseResults() {
        //var responseDiv = document.getElementById("WebPartManager1_AjaxWebPart2_
PetsServerResponse");
        var responseDivs = getElementsByTagNameAndEndingId("div", "AjaxResponse");
        var responseDiv = responseDivs[0];
        if (responseDiv.hasChildNodes()) {
            responseDiv.removeChild(responseDiv.childNodes[0]);
        }
        var responseText = document.createTextNode(xmlHttp.responseText);
        responseDiv.appendChild(responseText);
    }


    /*
    This function returns a list of DOM elements that have the tag name specified
and have an ID
    that ends in the value supplied.  The reason for this function is that ASP.NET
prepends control
    ID's with the webpart zone name and this function provides a handy way of
locating the our
    control of interest.
    tagName - DOM tag name to look for (e.g. div, select, p)
    endingID - String that the ID of the DOM elements should end with
    returns - list of DOM elements that have the tag name and have an id that ends
with value supplied
    */
    function getElementsByTagNameAndEndingId(tagName, endingId) {
        var tagItems = document.getElementsByTagName(tagName);
        var idValue;
        var idNode;
        var startPos;
        var returnElements = new Array(1);  // Typically only one item
        var j = 0;
        for (i = 0; i < tagItems.length; i++) {
            idNode = tagItems.item(i).attributes.getNamedItem("id");
            if (idNode != null) {
                idValue = idNode.nodeValue;
                startPos = idValue.indexOf(endingId);
                if (startPos > -1 && endingId == idValue.substring(startPos)) {
                    returnElements[j] = tagItems.item(i);
                    j++;
                }
            }
        }
        return returnElements;
    }
    </script>
```

The javascript above simply takes the value of a selected listbox item, passes it as a POST parameter to an URL, and then takes the response text and places it in a div. This is mostly standard AJAX methodology with two exceptions. The first is that the URL is grabbed from a hidden href so it will be properly transformed to a WSRP proxied URL. The second is that the URLs cannot contain query string parameters (like most AJAX requests) because they will not be properly appended by the WSRP Consumer.

Create the target page that the AJAX request will be sent to by creating a new ASPX page called AjaxPostDestination.aspx in the same directory as test.aspx. Paste the following code into the file:

```
<%@ Page Language="C#" AutoEventWireup="true" ValidateRequest="false" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<!-- NOTE that validaterequest is set to false.  This is the only way our
hand-rolled AJAX requests will be accepted. -->
<head id="Head1" runat="server">
    <title>Untitled Page</title>
    <script language="CS" runat="server">
        void Page_Load(object sender, System.EventArgs e){}
        protected override void Render(HtmlTextWriter writer)
        {
            // Take over the response writer and send only the results of the
selection
            // Turn cacheability off so that each request is viewed as "new"
            Response.Cache.SetCacheability(HttpCacheability.NoCache);
            // For this simple example, just send back plain text.  A more
sophisticated JSON or XML response could be set here.
        Response.ContentType = "text/plain";
            writer.Write("You selected " + Request.Params["petselection"]);
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

Note that the Master Page is not referenced; it would serve no purpose since that page is not displayed. Also note that the `ValidateRequest` attribute is set to "false". ASP.NET would throw a validation error without this attribute because the form is not being posted as expected (with more than a single parameter). Finally, note that caching is turned off. Since a query string parameter such as datetime cannot be added to the request, there is no way to make it unique.

Refresh the page where the portlet resides. The portlet should now display a list box with four pets listed in it. When you click on a pet name, you should see the pet selection displayed at the bottom of the portlet.

In summary, to use XMLHttpRequest in your code without using ASP.NET 3.5 AJAX controls, follow the rules below:

- Put the target URL for an XMLHttpRequest in a hidden anchor tag as an href (or use some other place like a hidden input field) and then grab the value of that href as the target URL to send in the XMLHttpRequest.
- If you need the response from the AJAX request to be transformed by the Portlet Content Transformation rulesets (for example, you have URLs you want proxied by the WSRP Consumer) add a "handle" query string argument to the target URL that references the handle you gave the portlet in wsrp-producer.xml.
- Do not use querystring parameters as a way of posting data. Use POST parameters only.
- Turn off caching in your target page so the request is not cached.

5.3 Adding an ASP.NET Application

If your portlet does not use preferences, you can use the Oracle WebCenter WSRP Producer for .NET to surface an ASP.NET application with WSRP. This tutorial adds an ASP.NET Web Form to the Oracle WebCenter WSRP Producer for .NET.

1. Create a very simple ASP.NET web form. For example, the test.aspx file below prints "Hello World" in an ASP.NET label control:

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <script language="CS" runat="server">
        void Page_Load(object sender, System.EventArgs e)
        {
            this.lblTest.Text = "Hello World";
        }
    </script>
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="lblTest" runat="server" Text=""></asp:Label>
    </div>
    </form>
</body>
</html>
```

2. Create a folder under <installdir>\wsrpdefault\portlets called "testASPX".
3. Copy the test.aspx file you created to the new folder.
4. Modify the aspx file to reference the Oracle WebCenter WSRP Producer for .NET Master Page as follows.

Note: It is important to understand that the Oracle WebCenter WSRP Producer for .NET uses a Master Page to render content. Your aspx file must reference the appropriate content placeholders of the Master Page as outlined below.

- Remove all DOM elements outside of the `<head>` and `<body>`.

- Replace the `<head>` element with the content placeholder identified by "wsrphead" and include any necessary child elements from the`<head>` element.

  ```
  <asp:Content ID="ContentHead1" ContentPlaceHolderID="wsrphead"
  Runat="Server">
  ```
- Replace the `<body>` and `<form>` elements with the content placeholder identified by "Main" and include any necessary child elements from the`<form>` element.

  ```
  <asp:Content ID="ContentBody1" ContentPlaceHolderID="Main" Runat="Server">
  ```

For example, a modified test.aspx file would look something like the following:

```
<%@ Page Language="C#"
MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
Page Test" %>
<asp:Content ID="ContentHead1" ContentPlaceHolderID="wsrphead" Runat="Server">
<script language="CS" runat="server">
        void Page_Load(object sender, System.EventArgs e)
        {
            this.lblTest.Text = "Hello World";
        }
    </script>
</asp:Content>
<asp:Content ID="ContentBody1" ContentPlaceHolderID="Main" Runat="Server">
<asp:Label ID="lblTest" runat="server" Text=""></asp:Label>
</asp:Content>
```

5. Create a new `<portlet>` entry in wsrp-producer.xml with a portlet handle named "testASPX". Remove the `<portlet-mode>` element for wsrp:edit since Web Forms do not support preferences.

6. Add the new "testASPX" portlet to your portal. You should see "Hello World" displayed.

In the above example, the C# code was placed directly in the file. If you want to use a code-behind .cs file, simply add that code to an \App_Code subfolder under <installdir>\wsrpdefault or add the compiled C# assembly to <installdir>\wsrpdefault\bin, and change the <%@ Page directive in the .aspx file to reference the assembly or class.

5.4 Adding a Help Page

To add a help file for any portlet, simply add a new `<portlet-mode>` element named wsrp:help that points to the help file. For example, to add a help file called testASPXhelp.htm to the previous example, the `<portlet-modes>` element would look like the following:

```
<portlet-modes>
   <portlet-mode>
    <name>wsrp:view</name>
    <url>/portlets/TestASPX/test.aspx</url>
   </portlet-mode>
   <portlet-mode>
    <name>wsrp:help</name>
    <url>/portlets/TestASPX/testASPXhelp.htm</url>
   </portlet-mode>
  </portlet-modes>
```

When you view the portlet in your portal, the help link for the portlet should bring you to the testASPXhelp.htm page.

5.5 Using Markup Transformation

The Oracle WebCenter WSRP Producer for .NET allows you to implement markup transformation to modify URLs and other static content before it is displayed in the browser.

5.5.1 Transforming Text

This tutorial provides a simplified example that modifies the "Hello World" text in Section 5.3, "Adding an ASP.NET Application" to read "Hello New World".

1. Open the <installdir>\wsrpdefault\wsrp-producer.xml file in a text editor.
2. Go to the `<portlet>` node with the handle "TestASPX" and add the following below the `<RuleSetRef>` element:

```
<RewriterConfig>
  <Rules>
   <RewriterRule>
    <LookFor>Hello World</LookFor>
    <ChangeToAbsolute>false</ChangeToAbsolute>
    <ChangeTo>Hello New World</ChangeTo>
    <ApplyTo>*</ApplyTo>
    <MakeResource>false</MakeResource>
   </RewriterRule>
  </Rules>
 </RewriterConfig>
```

   For details on ruleset elements in wsrp-producer.xml, see Section 2.2, "Portlet Content Transformation Rulesets."

3. Refresh the portal page that includes the testASPX portlet. The portlet should now display "Hello New World".

5.5.2 Creating a Custom Ruleset

This tutorial illustrates custom URL transformation by modifying the portlet from Section 5.2, "Adding a Web Part with ASP.NET AJAX Controls" to proxy the WebResource.axd and ScriptResource.axd files generated by the .NET framework when ASP.NET AJAX controls are added. (The Default ruleset defined at the top of the wsrp-producer.xml file ensures that URLs to web resource files generated by the .NET framework (*.axd) are made absolute, but are not proxied as WSRP resources. The practical reason for proxying .axd files would be a scenario in which WSRP Consumer portal users are not able to access URLs to your internal network.)

1. Open the <installdir>\wsrpdefault\wsrp-producer.xml file in a text editor.
2. Copy the entire Default ruleset (`<RuleSet id="Default">`) and rename the id to "axdproxy".
3. Locate the first `<RewriterRule>` in the new ruleset. In the `<ApplyTo>` element, add ", .axd".
4. Locate the second and third `<RewriterRule>` elements and remove ",.axd" from the `<ApplyTo>` elements under each.
5. Make a copy of the third `<RewriterRule>` (with `<LookFor>(src=\")'`). Change the `<ApplyTo>` element to include only ".axd" and change `<MakeResource>` to true.

   The finished ruleset should look something like the following:

```
<RuleSet id="axdproxy">
```

```
      <RewriterConfig>
        <Rules>
          <RewriterRule>
            <LookFor>(href=\")([^\"]+)</LookFor>
            <ChangeToAbsolute>true</ChangeToAbsolute>
            <ApplyTo>.aspx,.htm,.html,.axd</ApplyTo>
            <MakeResource>true</MakeResource>
          </RewriterRule>
          <RewriterRule>
            <LookFor>(href=\")([^\"]+)</LookFor>
            <ChangeToAbsolute>true</ChangeToAbsolute>
            <ApplyTo>.css</ApplyTo>
            <MakeResource>false</MakeResource>
          </RewriterRule>
          <RewriterRule>
            <LookFor>(src=\")([^\"]+)</LookFor>
            <ChangeToAbsolute>true</ChangeToAbsolute>
            <ApplyTo>.css,.js,.img,.png,.jpg,.gif</ApplyTo>
            <MakeResource>false</MakeResource>
          </RewriterRule>
          <RewriterRule>
            <LookFor>(src=\")([^\"]+)</LookFor>
            <ChangeToAbsolute>true</ChangeToAbsolute>
            <ApplyTo>.axd</ApplyTo>
            <MakeResource>true</MakeResource>
          </RewriterRule>
        </Rules>
      </RewriterConfig>
    </RuleSet>
```

6. Go to the `<portlet>` node with the handle "TestPreferences" and change the `<RuleSetRef>` element to reference the new axdproxy ruleset (`<RuleSetRef>axdproxy</RuleSetRef>`).
7. Save your changes to wsrp-producer.xml.
8. Refresh the portal page the includes the TestPreferences portlet. All URLs with .axd should now be proxied by the WSRP consumer portal. To confirm this, use a DOM inspection tool to search for .axd files; they will be transformed to a URL-encoded value that begins with the URL to the WSRP Consumer portal. To compare these URLs with un-proxied URLs, simply switch the `<RuleSetRef>` for the portlet back to "Default".

5.6 Using Portlet Styles

The Oracle WebCenter WSRP Producer for .NET does not require any special steps to apply skins and themes. Styles, themes, and skin management are delegated to the .NET framework. For example, to apply a particular WebCenter theme, simply place the source .css file in the \App_Themes folder and follow standard .NET rules to apply the style you want to a single control, a single portlet, or a set of portlets.

This simplified tutorial shows how to use built-in ASP.NET functionality to apply a style to all labels in all portlets. For more detailed information, see the Microsoft documentation on ASP.NET themes and skins.

1. Create a folder in <installdir>\wsrpdefault\App_Themes and name it "MyTheme".
2. Create a new file that contains the following line:
   `<asp:Label runat="server" ForeColor="white" BackColor="green"`

```
/>
```

3. Save the file in the \MyTheme folder as "my.skin".
4. Open the <installdir>\wsrpdefault\Web.config file in a text editor.
5. Locate the `<pages>` element and add the following attribute: `theme="MyTheme"`. This will make all ASP.NET pages use the themes defined in the "App_Themes\MyTheme" folder.
6. View any portlets that contain a label (for example, the portlets in the previous tutorials) and note that the background color is now green.

5.7 Using Interportlet Communication with JavaScript

The most direct way to pass data from one portlet to another is to use JavaScript. This tutorial shows a simple technique that uses one portlet as a "master" portlet from which a user selects an item. When an item is selected, data is passed to another portlet, which displays detailed information about the item.

First, create the "master" portlet. In Visual Studio, as part of a Web Application Project, create a new Web User Control and name it **MasterList.ascx**. For ease of deployment, don't put code in a separate file.

Add a label and a list box, but leave the file empty. The contents of the file should look like the following:

```
<%@ Control Language="C#" AutoEventWireup="true" %>
<asp:Label ID="lblHousingType" runat="server" Text="Select House
Type"></asp:Label>
<br/>
<asp:ListBox ID="listBoxHousing" runat="server" Height="80px" Width="171px">
    <asp:ListItem>Single Family</asp:ListItem>
    <asp:ListItem>Condominium</asp:ListItem>
    <asp:ListItem>Cooperative</asp:ListItem>
    <asp:ListItem>Tenancy in Common</asp:ListItem>
</asp:ListBox>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
    }
</script>
```

Add the following JavaScript at the bottom of the file:

```
<script type="text/javascript" language="javascript" >
// Set up an onchange event for listBoxHousing.
//This can also be done by adding onChange attribute to the asp:ListBox
var selectmenu;
var selectMenus = getElementsByTagNameAndEndingId("select", "listBoxHousing");
selectmenu = selectMenus[0];
selectmenu.onchange=function()
{
    // Update a hidden field in another portlet.  This is how information will be
passed
    // From one portlet to another.
    var i = selectmenu.selectedIndex;
    var text = selectmenu[i].value;
var portletFrame = getFrameFromSrcAttribute(window.parent.document, "DetailPage");
var portletContent = portletFrame.contentWindow;
portletContent.theForm.submitHousingField.value= text;
portletContent.theForm.submit();
}

/*
This function returns a list of DOM elements that have the tag name specified and
have an ID
that ends in the value supplied.  The reason for this function is that ASP.NET
prepends control
ID's with the webpart zone name and this function provides a handy way of locating
the our
control of interest.
tagName - DOM tag name to look for (e.g. div, select, p)
endingID - String that the ID of the DOM elements should end with
returns - list of DOM elements that have the tag name and have an id that ends
with value supplied
*/
function getElementsByTagNameAndEndingId(tagName, endingId) {
var tagItems = document.getElementsByTagName(tagName);
var idValue;
var idNode;
var startPos;
var returnElements = new Array(1);  // Typically only one item
var j = 0;
for (i = 0; i < tagItems.length; i++) {
idNode = tagItems.item(i).attributes.getNamedItem("id");
if (idNode != null) {
idValue = idNode.nodeValue;
startPos = idValue.indexOf(endingId);
if (startPos > -1 && endingId == idValue.substring(startPos)) {
returnElements[j] = tagItems.item(i);
j++;
}
}
}
return returnElements;
}
/* This function finds an iframe based on the src element
docSource - document element to search in for frame
srcSearch - string value that should be searched for in an iframe's src field
returns - iframe element
*/
function getFrameFromSrcAttribute(docSource, srcSearch) {
var iFrames = docSource.getElementsByTagName("iframe");
var srcNode;
var srcValue;
var returnFrame;
for (i = 0; i < iFrames.length; i++) {
srcNode = iFrames.item(i).attributes.getNamedItem("src");
if (srcNode != null) {
srcValue = srcNode.nodeValue;
startPos = srcValue.indexOf(srcSearch);
if (srcValue.indexOf(srcSearch) > -1) {
returnFrame = iFrames.item(i);
}
}
}
return returnFrame;
}
</script>
```

The code above takes the item selected from the listbox (a select DOM element), locates the "detail" portlet, and writes a value to a hidden field. The most important part of this code is how the second portlet is located by looking for the iframe that contains it based on text in the "src" attribute that uniquely identifies it. (In this sample, the unique identifier is the folder name under which the portlet will be placed, "DetailPage".)

The next step is to create the "detail" portlet. In Visual Studio, as part of a Web Application Project, create a new Web User Control and name it DetailPage.ascx. For ease of deployment, don't put code in a separate file. Add two ASP.NET labels and a hidden input field. The hidden field is where data from the "master" portlet will be placed. The contents of the file should look like the following:

```
<%@ Control Language="C#" AutoEventWireup="true" %>
<asp:Label ID="lblHousing" runat="server" Text="Housing Type Info:"></asp:Label>
<br/>
<asp:Label ID="lblHousingTypeInfo" runat="server" Text="None
selected"></asp:Label>
<!-- Use a hidden field to get data from another portlet -->
<input type="hidden" id="submitHousingField" name="submitHousingField" value=""/>
```

Add the following code to the bottom of the file.

```
<script runat="server">
 protected void Page_Load(object sender, EventArgs e)
 {
  if (Request.Params != null)
  {
   if (Request.Params["submitHousingField"] != null)
   {
    string housingType = Request.Params["submitHousingField"].ToString();
    // You'd probably look this up in a database in a real case, but simple switch
statement used for simplicity
    switch (housingType)
    {
     case "Single Family":
     {
      lblHousingTypeInfo.Text = "Single family homes consist of a structure and
the land.";
      break;
     }
     case "Condominium":
     {
      lblHousingTypeInfo.Text = "You own a condo, but the building housing the
condominium is maintained by the home owners association.";
      break;
     }
     case "Cooperative":
     {
      lblHousingTypeInfo.Text = "In a Cooperative, you own shares in the
corporation that owns the entire building.";
      break;
     }
     case "Tenancy in Common":
     {
      lblHousingTypeInfo.Text = "In a TIC, you own a share of the mortgage on the
entire building.";
      break;
     }
    default:
    {
     lblHousingTypeInfo.Text = "Unrecognized housing type selected.";
     break;
    }
```

```
        }
      }
    }
  }
</script>
```

The above code writes out values to the label based upon what is passed in a a form parameter from the hidden input field. If you look back at the MasterList.ascx file, you will see that it populates a hidden input field called "submitHousingField". (In a real implementation, the detail data would be taken from a database rather than hard-coded.)

The next set of steps set up the two Web User Controls you created in the previous steps and surfaces them as portlets using the Oracle WebCenter WSRP Producer for .NET.

1. Create the following folder: <installdir>\wsrpdefault\portlets\MasterList. Add a new file to the folder called default.aspx that contains the following code:

   ```
   <%@ Page Language="C#"
   MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
   Page Test" %>
   <%@ Register src= "~/portlets/MasterList/UserControl/MasterList.ascx"
   tagname="MasterList" tagprefix="uc1" %>
   <asp:Content ID="Content1" ContentPlaceHolderID="webpartcontrol"
   Runat="Server">
    <uc1:MasterList ID="MasterList1" runat="server" />
   </asp:Content>
   ```
2. Create the following folder: <installdir>\wsrpdefault\portlets\MasterList\UserControl and copy the MasterList.ascx file that you created to this folder.

3. Create the following folder: <installdir>\wsrpdefault\portlets\DetailPage. Add a new file to the folder called default.aspx that contains the following code:

   ```
   <%@ Page Language="C#"
   MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
   Page Test" %>
   <%@ Register src= "~/portlets/DetailPage/UserControl/DetailPage.ascx"
   tagname="DetailPage" tagprefix="uc1" %>
   <asp:Content ID="Content1" ContentPlaceHolderID="webpartcontrol"
   Runat="Server">
    <uc1:DetailPage ID="DetailPage1" runat="server" />
   </asp:Content>
   ```
4. Create the following folder: <installdir>\wsrpdefault\portlets\DetailPage\UserControl and copy the DetailPage.ascx file that you created to this folder.

5. Open <installdir>\wsrpdefault\wsrp-producer.xml in a text editor and add the following <portlet> elements to it. (For more information about the configuration file, see Chapter 3, "Managing Required Script Files.")

   ```
   <portlet>
    <handle>HousingType</handle>
    <supported-locale>en</supported-locale>
    <descriptions>
     <description lang="en">Housing Type</description>
    </descriptions>
    <display-names>
     <display-name lang="en">Housing Type</display-name>
    </display-names>
    <titles>
     <title lang="en">Housing Type</title>
   ```

```
  </titles>
  <short-titles>
   <short-title lang="en">Housing Type</short-title>
  </short-titles>
  <supports>
   <portlet-modes>
    <portlet-mode>
     <name>wsrp:view</name>
     <url>/portlets/MasterList/default.aspx</url>
    </portlet-mode>
   </portlet-modes>
   <mime-type>text/html</mime-type>
  </supports>
 </portlet>

 <portlet>
  <handle>HousingDetail</handle>
  <supported-locale>en</supported-locale>
  <descriptions>
   <description lang="en">Housing Detail</description>
  </descriptions>
  <display-names>
   <display-name lang="en">Housing Detail</display-name>
  </display-names>
  <titles>
   <title lang="en">Housing Detail</title>
  </titles>
  <short-titles>
   <short-title lang="en">Housing Detail</short-title>
  </short-titles>
  <supports>
   <portlet-modes>
    <portlet-mode>
     <name>wsrp:view</name>
     <url>/portlets/DetailPage/default.aspx</url>
    </portlet-mode>
   </portlet-modes>
   <mime-type>text/html</mime-type>
  </supports>
 </portlet>
```

6. Register the WSRP producer by pointing your WSRP Consumer at the WSDL URL that is in the shortcut Start | Programs | Oracle Application Accelerator for .NET | WSRP 1.0 WSDL. (Refer to the product documentation for your WSRP Consumer portal for details on how to register a WSRP producer WSDL.)
7. Add the "HousingType" and "HousingDetail" portlets (these are the names defined in the wsrp-producer.xml file) to a single page in your portal. Whenever you click on a housing type in one portlet, details about the housing type should appear in the other portlet.

This tutorial shows just one simple way that you can add interportlet communication functionality to portlets.

5.8 Adding User Profile Data

This tutorial creates a Web Part that uses user profile data. This example creates the Web Part as a class that inherits from System.Web.UI.WebControls.WebParts.WebPart. You could also create a Web User

Control to perform the same function. See the Microsoft documentation for more details on both options.

IMPORTANT: This note is very important for any WSRP Consumer that uses preferences as defined in the WSRP specification (including Oracle WebCenter).

ASP.NET supports only one level of nesting for profile properties (properties can only be nested under a <group>), but many of the UserProfile items defined in the specification are nested several levels deep. For example, a user's business email would be defined as businessInfo/online/email and a user's home mobile phone extension would be defined as homeInfo/telecom/mobile/number in the WSRP specification. To account for this, the Oracle WebCenter WSRP Producer for .NET uses the "_" character as a replacement for nested objects. This is shown in the steps that follow where businessInfo/online/email is treated as businessInfo_online.email in the C# code, but referenced in Web.config as:

```
<group name="businessInfo_online">
  <add name="email" defaultValue="John.Doe@acme.com"/>
</group>
```

Oracle WebLogic Portal follows the same single level of nesting rules that ASP.NET uses, so this issue does not apply.

---

1. Create a new C# class in Visual Studio and copy in the following code:

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using System.Configuration;

using System.Web;
using System.Web.Profile;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace TestUserProfile
{
    public class UserProfileWebPart : WebPart
    {

        Label lblEmailWLP;
        Label lblEmailWSRP;
        protected override void CreateChildControls()
        {

            // Create a label and assign a user profile value.  Try the WLP
format and then the standard WSRP format.
            lblEmailWLP = new Label();
            lblEmailWLP.ID = "lblEmailWLP";
            lblEmailWLP.Text = "WLP profile email: " +
this.Context.Profile["GroupSpace.email"].ToString();
            this.Controls.Add(lblEmailWLP);
            HtmlGenericControl br1 = new HtmlGenericControl("br");
            this.Controls.Add(br1);

            lblEmailWSRP = new Label();
            lblEmailWSRP.ID = "lblEmailWSRP";
            lblEmailWSRP.Text = "WSRP profile email: " +
this.Context.Profile["businessInfo_online.email"].ToString();
            this.Controls.Add(lblEmailWSRP);
            HtmlGenericControl br2 = new HtmlGenericControl("br");
            this.Controls.Add(br2);
        }
    }
}
```

2. Compile this code into an assembly called "TestUserProfile" and copy the assembly to <installdir>\wsrpdefault\bin.
3. Open <installdir>\wsrpdefault\Web.config and update the profile properties so that those referenced by the code are used. Note that in the C# code, profile properties are specified in group.name format.

   Locate the `<properties>` element and replace it with the following:

```
      <properties>
        <group name="businessInfo_online">
          <add name="email" defaultValue="John.Doe@acme.com"/>
        </group>
        <group name="GroupSpace">
          <add name="email" defaultValue="John.Doe@acme.com"/>
        </group>
      </properties>
```

Note: For this example to work, there must be a default value. In WebCenter or other WSRP consumers that use the WSRP specification UserProfile items the GroupSpace.email property will not be available. WLP uses UserProfile extensions to pass items, so you could not create a nested businessInfo/online/email profile property.

4. Create a folder under <installdir>\wsrpdefault\portlets called "TestUserProfile".
5. Copy the default.aspx file from <installdir>\wsrpdefault\portlets\ASPNET_AJAX_sample to your new \TestUserProfile folder.
6. Update the <installdir>\wsrpdefault\portlets\TestPreferences\default.aspx file to reference the new TestUserProfile assembly instead of the Simple_Date_Time control.
   - Replace "src" with "assembly" and reference the TestUserProfile assembly you created.
   - Replace "tagname" with "namespace" and reference the TestUserProfile namespace used by the class.
   - Add the class name after the ":" following the tagprefix under the content placeholder.

   The resulting file will look something like the following:

```
<%@ Page Language="C#"
MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
Page Test" %>
<%@ Register assembly="TestUserProfile" namespace="TestUserProfile"
```

```
tagprefix="uc1" %>
 <asp:Content ID="Content1" ContentPlaceHolderID="webpartcontrol"
Runat="Server">
 <uc1:UserProfileWebPart ID="UserProfileWebPart1" runat="server" />
</asp:Content>
```

7. Update the <installdir>\wsrpdefault\wsrp-producer.xml file to include your portlet AND the profile properties you are using.

   - Make a copy of an existing `<portlet>` node and its contents under the `<portlets>` element.
   - Update the handle, display name, description, title, and short title elements for the new portlet.
   - Update the `<url>` node under each of the `<portlet-mode>` elements to point to the new portlet. (The wsrp:edit and wsrp:help modes are not necessary.)
   - Add the profile properties referenced in the portlet to the `<profile-item>` elements under the `<user-profile>` element.

   When finished, the new `<portlet>` node should look like the following:

```
<portlet>
 <handle>TestUserProfile</handle>
 <supported-locale>en,fr</supported-locale>
 <descriptions>
  <description lang="en">Test UserProfile</description>
  <description lang="fr">Test UserProfile</description>
 </descriptions>
 <display-names>
  <display-name lang="en">Test UserProfile</display-name>
  <display-name lang="fr">Test UserProfile</display-name>
 </display-names>
 <titles>
  <title lang="en">Test UserProfile</title>
  <title lang="fr">Test UserProfile</title>
 </titles>
 <short-titles>
  <short-title lang="en">Test UserProfile</short-title>
  <short-title lang="fr">Test UserProfile</short-title>
 </short-titles>
 <supports>
  <portlet-modes>
   <portlet-mode>
    <name>wsrp:view</name>
    <url>/portlets/TestUserProfile/default.aspx</url>
   </portlet-mode>
  </portlet-modes>
  <mime-type>text/html</mime-type>
 </supports>
 <user-profile>
  <profile-item>businessInfo/online/email</profile-item>
  <profile-item>GroupSpace/email</profile-item>
 </user-profile>
</portlet>
```

8. The portlet should now be available in your WSRP Consumer. (You should have already registered the WSRP producer as explained in Section 5.1, "Adding a Web Part as a Portlet.") Add the portlet to your portal. You should the two labels showing the default values for the profile properties from Web.Config.

9. Configure your WSRP Consumer to send the email preference according to your portal documentation.

   For Oracle WebLogic Portal, use the steps below. (The default WLP Data Sync has a PropertySet called "GroupSpace" and one of the properties in this PropertySet is called "email".)

   a. Log into the WLP Administration Console.
   b. Select **Users, Groups, and Roles** | **User Management**.
   c. Click on the user you have been using to test Web Parts.
   d. Click on the User Profile tab.
   e. Select the GroupSpace PropertySet in the dropdown menu. (If this PropertySet does not exist, create a new PropertySet called "GroupSpace" and populate it with a "Single, Unrestricted" property called "email".)
   f. Edit the "email" property (enter a value).
   g. Log in to Oracle WebLogic Portal with the user you edited and view the UserProfile Web Part portlet to see your changes.

5.9 Setting Up Security Tokens

By default, the Oracle WebCenter WSRP Producer for .NET pulls the XML containing a `<wsse:Security>` element defined in a SOAP header and puts it into the HttpContext.Items collection. The token will only be available for the duration of the ASP.NET request and is identified by the ASP.NET session ID.

Due to the wide number of possibilities that the WS-Security specification allows for structuring security tokens the Oracle WebCenter WSRP Producer for .NET installation includes the source code for its default security implementation in <installdir>\wsrpdefault\src. For the default SecurityTokenManager implementation, see the files in the \token subfolder. For the default request filter that is applied specifically for WebCenter WSS 1.0 tokens before the tokens are passed to the default SecurityTokenManager, see the files in the \util subfolder.

This section provides several tutorials illustrating how to access security tokens in a number of ways.

- Section 5.9.1, "Displaying a Token in a Web Part"
- Section 5.9.2, "Oracle WebLogic Portal with Default SAMLCredentialMapperV2"
- Section 5.9.3, "Oracle WebLogic Portal with Custom Signing Key"
- Section 5.9.4, "UNT for WebLogic Portal"
- Section 5.9.5, "SAML for Oracle WebCenter"
- Section 5.9.6, "UNT for Oracle WebCenter"

5.9.1 Displaying a Token in a Web Part

This tutorial creates a simple Web Part that displays a security token. The Web Part created is generic enough that it can be used for looking at any type of wsse:security token (including SAML from WLP or UNT from WebCenter). The Web Part is created as a class that inherits from System.Web.UI.WebControls.WebParts.WebPart (you could provide the same functionality with a Web User Control).

1. Create a new C# class in Visual Studio. Make sure to add a reference to the WSE assembly component Microsoft.Web.Services2 or the sample code will not compile. Copy in the following code:

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Configuration;

using System.Web;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml;

using Microsoft.Web.Services2.Security.Tokens;

namespace SecurityTokenWebPart
{
    public class SecurityTokenWebPart : WebPart
    {
        Label lblSamlHeader;
        Label lblUntHeader;
        Label lblSecurityHeader;
        Label lblSamlToken;
        Label lblUntToken;
        Label lblSecurityToken;
        protected override void CreateChildControls()
        {
            // Create text label for SAML token
            lblSamlHeader = new Label();
            lblSamlHeader.ID = "lblSamlHeader";
            lblSamlHeader.Text = "SAML token:";
            this.Controls.Add(lblSamlHeader);
            HtmlGenericControl br1 = new HtmlGenericControl("br");
            this.Controls.Add(br1);

            // Retrieve the SAML security token associated with this session,
if available
            lblSamlToken = new Label();
            lblSamlToken.ID = "lblSamlToken";
            Object item = HttpContext.Current.Items[Page.Session.SessionID +
".SAML_TOKEN"];
            if (item != null)
            {
                lblSamlToken.Text =
HttpUtility.HtmlEncode(HttpUtility.UrlDecode(item.ToString()));
            }
            else
            {
                lblSamlToken.Text = "SAML token unavailable";
            }
            this.Controls.Add(lblSamlToken);
            HtmlGenericControl br2 = new HtmlGenericControl("br");
            this.Controls.Add(br2);


            // Create text label for UNT token
            lblUntHeader = new Label();
            lblUntHeader.ID = "lblUntHeader";
            lblUntHeader.Text = "UNT token:";
            this.Controls.Add(lblUntHeader);
            HtmlGenericControl br3 = new HtmlGenericControl("br");
            this.Controls.Add(br3);

            // Retrieve the UNT security token associated with this session, if
available
            lblUntToken = new Label();
            lblUntToken.ID = "lblUntToken";
            item = HttpContext.Current.Items[Page.Session.SessionID +
".USERNAME_TOKEN"];
            if (item != null)
            {
                lblUntToken.Text =
HttpUtility.HtmlEncode(HttpUtility.UrlDecode(item.ToString()));
            }
            else
            {
                lblUntToken.Text = "Username token (UNT) unavailable";
            }
            this.Controls.Add(lblUntToken);
            HtmlGenericControl br4 = new HtmlGenericControl("br");
            this.Controls.Add(br4);

            // Create text label for security tokens
            lblSecurityHeader = new Label();
            lblSecurityHeader.ID = "lblSecurityHeader";
            lblSecurityHeader.Text = "Security token(s):";
            this.Controls.Add(lblSecurityHeader);
            HtmlGenericControl br5 = new HtmlGenericControl("br");
            this.Controls.Add(br5);

            // Retrieve all the security tokens associated with this session,
if available
            lblSecurityToken = new Label();
            lblSecurityToken.ID = "lblSecurityToken";
            item = HttpContext.Current.Items[Page.Session.SessionID +
".TOKENS"];
            if (item != null)
            {
                SecurityTokenCollection tokens = item as
SecurityTokenCollection;
                if (tokens.Count > 0)
                {
                    foreach (SecurityToken token in tokens)
                    {
                        XmlElement xe = token.GetXml(new XmlDocument());
                        lblSecurityToken.Text +=
HttpUtility.HtmlEncode(xe.OuterXml);
                    }
                }
                else
                {
                    lblSecurityToken.Text = "Security token unavailable";
                }
            }
            else
            {
                lblSecurityToken.Text = "Security token unavailable";
            }
            this.Controls.Add(lblSecurityToken);
        }
    }
```

2. Compile this code into an assembly named "ViewSecurityToken".
3. Copy the assembly into <installdir>\wsrpdefault\bin.
4. Create a folder under <installdir>\wsrpdefault\portlets called "ViewSecurityToken".
5. Copy the default.aspx file from <installdir>\wsrpdefault\portlets\ASPNET_AJAX_sample to your new \ViewSecurityToken folder.
6. Update the <installdir>\wsrpdefault\portlets\ViewSecurityToken\default.aspx file to reference the new TestUserProfile assembly instead of the Simple_Date_Time control.
   - Replace "src" with "assembly" and reference the ViewSecurityToken assembly you created.
   - Replace "tagname" with "namespace" and reference the SecurityTokenWebPart namespace used by the class.
   - Add the class name after the ":" following the tagprefix under the content placeholder.

   The resulting file will look something like the following:

```
<%@ Page Language="C#"
MasterPageFile="~/portlets/Resources/MasterPages/WSRP.Master" Title="Content
Page Test" %>
<%@ Register assembly="ViewSecurityToken" namespace="SecurityTokenWebPart"
tagprefix="uc1" %>
 <asp:Content ID="Content1" ContentPlaceHolderID="webpartcontrol"
Runat="Server">
 <uc1:SecurityTokenWebPart ID="SecurityTokenWebPart1" runat="server" />
</asp:Content>
```

7. Update the <installdir>\wsrpdefault\wsrp-producer.xml file to include your portlet.
   - Make a copy of an existing `<portlet>` node and its contents under the `<portlets>` element.
   - Update the handle, display name, description, title, and short title elements for the new portlet.
   - Update the `<url>` node under each of the `<portlet-mode>` elements to point to the new portlet. (The wsrp:edit and wsrp:help modes are not necessary.)

   When finished, the new `<portlet>` node should look like the following:

```
<portlet>
 <handle>ViewSecurityToken</handle>
 <supported-locale>en</supported-locale>
 <descriptions>
  <description lang="en">View SecurityToken</description>
 </descriptions>
 <display-names>
  <display-name lang="en">View SecurityToken</display-name>
 </display-names>
 <titles>
```

```
  <title lang="en">View SecurityToken</title>
 </titles>
 <short-titles>
  <short-title lang="en">View SecurityToken</short-title>
 </short-titles>
 <supports>
  <portlet-modes>
   <portlet-mode>
    <name>wsrp:view</name>
    <url>/portlets/ViewSecurityToken/default.aspx</url>
   </portlet-mode>
  </portlet-modes>
  <mime-type>text/html</mime-type>
 </supports>
</portlet>
```

8. The portlet should now be available in your WSRP Consumer. (You should have already registered the WSRP producer as explained in Section 5.1, "Adding a Web Part as a Portlet.") Add the portlet to your portal. Assuming that you have not set up SAML or UNT tokens you should just see "Security token unavailable" when you view the portlet.

5.9.2 Oracle WebLogic Portal with Default SAMLCredentialMapperV2

The Oracle WebCenter WSRP Producer for .NET is configured to request SAML tokens by default. Oracle WebLogic Portal is set up with a SAML Credential Mapper version (SAMLCredentialMapperV2) that is compliant with this version of the Oracle WebCenter WSRP Producer for .NET by default. This tutorial illustrates how to view SAML tokens from Oracle WebLogic Portal.

1. Log in to the WebLogic Server hosting your Oracle WebLogic Portal.
2. Go to Security Realms | myrealm | Providers | Credential Mapping.
3. Click on SAMLCredentialMapper. (The description should include "Security Assertion Markup Language v1.1" and the version should be "2.0".)
4. Confirm that this is the default setup by clicking the Provider Specific tab and noting that the Signing Key Alias is "wsrpconsumerrsa".
5. Click the Management tab and on the "Relying Party" link named "rp_00001".
6. Make sure that both Enabled and Sign Assertions are true (they should be true by default).
7. Select **Include Key Info**.
8. Click **Save**.
9. Restart Oracle WebLogic Server (even if WLS indicates a restart is not necessary).
10. Log in to Oracle WebLogic Portal and view the portlet you created Section 5.9.1, "Displaying a Token in a Web Part." The portlet should display a raw SAML security token.

5.9.3 Oracle WebLogic Portal with Custom Signing Key

The previous section used the default "wsrpconsumerrsa" Signing Key Alias. In a typical SAML set up, you would use your own private key to sign SAML assertions. This section explains how to tailor the steps defined in Oracle WebLogic Portal documentation for creating SAML assertion to account for a WSRP producer created by the Oracle WebCenter WSRP Producer for .NET.

When configuring a Relying Party for the Oracle WebCenter WSRP Producer for .NET, follow the guidelines below:

- Create the Relying Party with "WSS/Sender-Vouches". (You can ignore the "Source Site Federation Services" portion of the WebLogic documentation as long as you are using WSS/Sender-Vouches.)
- Make sure that Sign Assertions and Include Keyinfo are selected.
- If the Relying Party will not use a value of "default" (all parties rely on this SAML credential mapper), be sure to put the full URL to the .asmx file defined in your WSRPService.wsdl.

5.9.4 UNT for WebLogic Portal

This tutorial illustrates how to view UNT tokens with a username and password from Oracle WebLogic Portal. The first two steps make sure the Oracle WebCenter WSRP Producer for .NET is set up to request UNT with username and password when it is registered, and the rest of the steps configure WLP to send UNT tokens with username and password.

1. Open the `<`*`installdir`*`>\wsrpdefault\wsdl\1.0\WSRPService.wsdl` file in a text editor.
2. Replace the `wsp:Policy` element that references `SAMLAuth.xml` with the following:

```
  <wsp:Policy s1:Id="UNTAuth.xml">
    <wsp:All>
      <wssp:Identity>
        <wssp:SupportedTokens>
          <wssp:SecurityToken
TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-tok
en-profile-1.0#UsernameToken">
            <wssp:UsePassword
Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-pr
ofile-1.0#PasswordText"/>
          </wssp:SecurityToken>
        </wssp:SupportedTokens>
      </wssp:Identity>
    </wsp:All>
  </wsp:Policy>
```

3. Open the `<`*`installdir`*`>\wsrpdefault\wsdl\1.0\wsrp_v1_bindings.wsdl` file in a text editor.
4. Replace all instances of "SAMLAuth.xml" with "UNTAuth.xml". You should see the following under the <wsdl:input> elements with the names "getMarkup" and "performBlockingInteraction":

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

Note: The PolicyReference URI value is the #Id of the wsp:Policy defined for UNT in <installdir>\wsrpdefault\wsdl\1.0\WSRPService.wsdl.

5. Log in to the WebLogic Server hosting your Oracle WebLogic Portal.

6. Go to Security Realms | myrealm | Credential Mappings.
7. Click **New** and enter settings that point to the markup endpoints. Leave the defaults except the following:
   - Protocol: http
   - Remote Host: <host name> (e.g., myproducer.mycompany.com)
   - Remote Port: <host port> (e.g., 8678)
   - Path: /wsrpdefault/wsdl/1.0/WSRPBaseService.asmx (This is the path after the host and port in the URL. Be sure to begin the path with "/".)
8. Before clicking **Next**, confirm that the values are correct. You should get a response from http://<host name>:<host port>/wsrpdefault/wsdl/1.0/WSRPBaseService.asmx.
9. Click **Next**.
10. Enter user information:
    - Local user: <A user registered with WebLogic Server, for example, the "weblogic" user>
    - Remote user: testuser <The user to send to the producer when the "Local user" is logged in>
    - Remote password: testpassword <The password to send to the producer when the "Local user" is logged in>
11. Click **Finish**.
12. Restart Oracle WebLogic Server (even if WLS indicates a restart is not necessary).
13. Register your producer with Oracle WebLogic Portal under a new name so that portlets now ask for UNT tokens instead of SAML tokens.
    1. Log in to the WebLogic Portal administration console and remove the portlet you registered inSection 5.9.1, "Displaying a Token in a Web Part."
    2. Create a new WSRP Producer in the administration console pointing to the same WSRPService.wsdl you used to register the first portlet.
    3. Add the same portlet again.
    4. Log in to WebLogic Portal with the "Local user" you entered in the Credential Mapper and view the portlet. The portlet should display a raw UNT security token.

5.9.5 SAML for Oracle WebCenter

There are several different SAML token profiles supported by Oracle WebCenter. The Oracle WebCenter WSRP Producer for .NET supports all of the WSS 1.0 options, but does not support WSS 1.1. This tutorial uses the SAML profile that is the most similar to the default provided by WLP (see Section 5.9.3, "Oracle WebLogic Portal with Custom Signing Key"), WSS 1.0 SAML Token with Message Identity. See the WebCenter documentation on securing a WSRP Producer for details on the security token options available.

This tutorial describes the process for setting up this token profile in Enterprise Manager. If you register WSRP producers using WLST, register your producer with the SAML_TOKEN_WITH_MSG_INTEGRITY token type and other arguments that correspond to Enterprise Manager settings described below. You should already have completed the tutorial inSection 5.9.1, "Displaying a Token in a Web Part" before beginning this tutorial.

1. Create a private key pair on the WebCenter host machine using the java keytool. Set the key algorithm to "rsa" and the key size to 1024. Refer to WebCenter or Java documentation on generating key pairs. (An example command is C:\Oracle\Middleware\jdk160_20\bin\keytool -genkeypair -keystore C:\Oracle\Middleware\keystores\mykeystore.jks -storepass mykeystorepass -keyalg rsa -keysize 1024 -alias wckey -keypass wckeypass.)
2. In Enterprise Manager, go to the **Add New Portlet Producer** page and select **WSRP Producer**.
3. Enter the URL to the same WSDL you used in Section 5.9.1, "Displaying a Token in a Web Part".
4. In the security section, select the token profile **WSS 1.0 SAML Token with Message Integrity**.
5. Choose the **custom** option and define an Issuer Name. This can be any name you wish.
6. Enter values in the **Keystore** section of the page based on the values you used to generate the key with the keytool. (Leave the encryption key values empty; they are not required because encryption is not used.)

Keystore value	Keytool argument
Store Path	-keystore
Password	-storepass
Signature Key Alias	-keyalias
Signature Key Password	-keypass

7. Once the producer is registered, add the ViewSecurityToken portlet to a WebCenter Space. If you are logged in (not anonymous) to WebCenter, the portlet should show a SAML token with your user name in the Assertion.

Note on SAML with Message Protection: If you choose to use WSS 1.0 SAML with Message Protection, you must create a private key in the Windows certificate store and import a public key certificate into the java keystore on your WebCenter Spaces host machine. Once you have imported the certificate, enter the keyalias you used during the keytool import command as the **Recipient Alias**. This way the imported certificate will be used to encrypt the message to the producer.

The process for managing the windows certificate store is explained in the next section, Section 5.9.6, "UNT for Oracle WebCenter". The steps are the same for SAML, except you must select the WSS 1.0 SAML with Message Protection option instead of WSS 1.0 Username Token with Password.

5.9.6 UNT for Oracle WebCenter

There are two different Username token (UNT) profiles supported by Oracle WebCenter. The Oracle WebCenter WSRP Producer for .NET supports both of these options. This tutorial describes working with the UNT profile that is the most similar to the default provided by WLP (see Section 5.9.4, "UNT for WebLogic Portal"), WSS 1.0 Username Token with Password. See the WebCenter documentation on securing a WSRP Producer for more details on the security token options available.

This tutorial describes the process for setting up this token profile in Enterprise Manager. If you register WSRP producers using WLST, register your producer with the USERNAME_WITH_PASSWORD token type and other arguments that correspond to Enterprise Manager settings described below. You should already have completed the tutorial inSection 5.9.1, "Displaying a Token in a Web Part" before beginning this tutorial.

1. Add a private key in the Windows certificate store on the Windows machine hosting your WSRP Producer. The following steps show you how to create a private key in LocalMachine | TrustedPeople Store and LocalMachine | Personal Store. Refer to your Microsoft documentation for more information on the Windows certificate store.

    a. Locate the makecert.exe tool. If this tool is not present, download the Microsoft Windows SDK for Windows and .NET Framework 3.5 SP1.

    b. Open a command window that has the makecert.exe PATH dependencies available. If Visual Studio 2008 is present, you can launch a Visual Studio 2008 Command Prompt from the Start Menu. If the Windows SDK is present, you can launch a command shell from under the Microsoft Windows SDK folder in the Start Menu.

    c. Create a certificate with the algorithm set to sha1. An example command is: makecert.exe -a sha1 -n CN=DotNetWSRPProducer -sr LocalMachine -ss My -sky exchange -sk DotNetWSRPProducer.

2. Import the key certificate into the TrustedPeople Store and Personal Store of the Windows certificate store. Use the same command window where you ran makecert.exe to run certmgr.exe. An example command is: certmgr.exe -add -c -n DotNetWSRPProducer -s -r LocalMachine My -s -r LocalMachine TrustedPeople

3. Grant security rights to the certificate for the user running the ASP.NET processes for the IIS application running the Oracle WebCenter WSRP Producer for .NET (by default this is the Network Service user).

    - If you installed the administrator version of WSE 2.0 SP3, use the X509 certificate tool available in the Start Menu, open the certificate from the correct store, view the key file properties, and make sure Network Service has full control.

    - If you do not have the X509 certificate tool, you can still set proper permissions on the certificate you created by going to %SystemDrive%\ProgramData\Microsoft\Crypto\RSA\MachineKeys, finding the most recent file (these are system files so you must configure Windows Explorer to show hidden files), and granting the Network Service user full control.

4. Locate the certificate in the store.

    a. Bring up mmc.exe from the command line.

    b. From the menu bar, select File | Add/Remove Snap-in.

    c. Select certificates.

    d. Select Computer Account (to view the LocalMachine contents).

    e. Go to the Personal section of the store and locate the certificate you added.

5. Export the certificate.

   a. Right-click on your certificate and select All Tasks | Export...

   b. In the wizard that pops up, export the public key as a DER encoded binary (DER is X509 format that java keytool likes) with a file name of your choice (e.g. dnaakey.der).

6. Copy the .der file over to the WebCenter Spaces box and import the key using the java key tool. An example command is: C:\Oracle\Middleware\jdk160_20\bin\keytool -genkeypair -keystore C:\Oracle\Middleware\keystores\mykeystore.jks -import -storepass mykeystorepass -keyalg rsa -keysize 1024 -alias dnaakey -keypass dnaakeypass -file C:\Oracle\Middleware\keystores\dnaakey.der

7. Create a private key pair on the WebCenter host machine using the java keytool. Set the key algorithm to "rsa" and the key size to 1024. If you already completed Section 5.9.5, "SAML for Oracle WebCenter," use the same key for this tutorial. Refer to your WebCenter or Java documentation on generating key pairs. An example command is: C:\Oracle\Middleware\jdk160_20\bin\keytool -genkeypair -keystore C:\Oracle\Middleware\keystores\mykeystore.jks -storepass mykeystorepass -keyalg rsa -keysize 1024 -alias wckey -keypass wckeypass

8. In Enterprise Manager, go to the **Add New Portlet Producer** page and select **WSRP Producer**.

9. Enter the URL to the same WSDL you used in Section 5.9.1, "Displaying a Token in a Web Part."

10. In the security section, select the token profile **WSS 1.0 Username Token with Password**.

11. Choose the **custom** option.

12. In the **Associated External Application** section choose **Create New..**. You will be brought to a new page where you can create a new external application. Give the application a name and click OK. This tells WebCenter to prompt end users to enter the login credentials they want to pass to the WSRP Producer portlet.

13. Once you are returned to the Add New Portlet Producer page, enter values in the **Keystore** section of the page based on the values you used to generate the key with the keytool. (Leave the encryption key values empty; they are not required because encryption is not used.)

Keystore value	Keytool argument
Store Path	-keystore
Password	-storepass
Signature Key Alias	-keyalias
Signature Key Password	-keypass
Recipient Alias (Note: This value comes from importing the certificate (step 6), not generating the key.)	-keyalias

14. Once the producer is registered, add the ViewSecurityToken portlet to a WebCenter Space. The first time you view the portlet you should see an "Update login information" link. Click this link and provide a user name and password.

This will be the user name and password that is sent to the portlet in the Username and password token (UNT). Once you enter this information the portlet should show a token with the user name and password you entered.

5.10 Using Oracle WebCenter Features

The following features are included specifically for Oracle WebCenter. Since these features are based on WSRP standards there is a possibility they could be applied to any WSRP Consumer.

5.10.1 Using Customization in Oracle WebCenter

This tutorial shows how to enable the Customize option for Web Parts surfaced in an Oracle WebCenter GroupSpace. This example starts with the "TestPreferences" portlet described in Section 5.1, "Adding a Web Part as a Portlet."

Open <installdir>\wsrpdefault\wsrp-producer.xml in a text editor. Locate the TestPreferences portlet and add a new <portlet-mode> section under the <portlet-modes> element with a name of urn:javax:portlet:mode:custom:edit_defaults. The <supports> element for the portlet with the new value should look something like the following:

```
<supports>
  <portlet-modes>
   <portlet-mode>
    <name>wsrp:view</name>
    <url>/portlets/TestPreferences/default.aspx</url>
   </portlet-mode>
   <portlet-mode>
    <name>wsrp:edit</name>
    <url>/portlets/TestPreferences/default.aspx</url>
   </portlet-mode>
   <portlet-mode>
    <name>urn:javax:portlet:mode:custom:edit_defaults</name>
    <url>/portlets/TestPreferences/default.aspx</url>
   </portlet-mode>
  </portlet-modes>
  <mime-type>text/html</mime-type>
 </supports>
```

Add the TestPreferences portlet to an Oracle WebCenter Space, and while in Oracle Composer, select the Customize option (wrench icon).

5.10.2 Using Oracle WebCenter Parameters

Oracle WebCenter parameters are passed to WSRP 2.0 producers using the navigationalParameters defined in the WSRP 2.0 specification. This tutorial shows how to pass WebCenter parameters to a portlet.

In Visual Studio, open the TestPreferences.ascx Web User Control you created in Section 5.1, "Adding a Web Part as a Portlet". Add two labels called lblParameter1 and lblParameter2 to the bottom of the form.

```
<br />
<asp:Label ID="lblParameter1" runat="server" Text=""></asp:Label></br>
<asp:Label ID="lblParameter2" runat="server" Text=" "></asp:Label>
```

Add the following code to the bottom of the Page_Load method:

```
Dictionary<string, string> ParameterCollection =
HttpContext.Current.Items["NavigationalParameters"] as Dictionary<string, string>;
lblParameter1.Text = ParameterCollection["TestParameter1"];
lblParameter2.Text = ParameterCollection["TestParameter2"];
```

Test that this compiles. In particular, make sure that you have System.Collections.Generic referenced. Update your TestPreferences.ascx and TestPreferences.ascx.cs (and designer .cs file if you created one of these) under the App_Code directory.

The following steps are required to inform the WebCenter WSRP Consumer of the existence of these parameters. Open <installdir>\wsrpdefault\wsrp-producer.xml in a text editor and locate the TestPreferences <portlet> element. Above the <supports> element add the following XML:

```
<navigational-parameters>
  <parameter>
    <identifier>TestParameter1</identifier>
  </parameter>
  <parameter>
    <identifier>TestParameter2</identifier>
  </parameter>
</navigational-parameters>
```

Note that the identifier names match the names pulled from the navigational parameters collection in the Page_Load method described in an earlier step.

In an Oracle WebCenter Personal or GroupSpace page, open the WebCenter page editor. Add the TestPreferences portlet and click the Edit button. On the parameters tab you should see the two parameters you defined, TestParameter1 and TestParameter2. Enter a value for each parameter and click Apply and Ok. You should see the values you entered at the bottom of the portlet.

Note: The Oracle WebCenter WSRP Consumer only utilizes the identifier portion of the navigationalParameter. However, if you have a different WSRP Consumer for which you wish to use navigationParameters you may also assign localized values for the description, label, and hint in your wsrp-producer.xml. For example:

```
<navigational-parameters>
 <parameter>
  <identifier>TestParameter1</identifier>
  <description>
   <parameter-description
lang="en">parameter1-en</parameter-description>
   <parameter-description
lang="fr">parameter1-fr</parameter-description>
  </description>
  <label>
   <parameter-label lang="en">parameter1-label-en</parameter-label>
   <parameter-label lang="fr">parameter1-label-fr</parameter-label>
  </label>
  <hint>
   <parameter-hint lang="en">parameter1-hint-en</parameter-hint>
   <parameter-hint lang="fr">parameter1-hint-fr</parameter-hint>
  </hint>
 </parameter>
</navigational-parameters>
```
