BEA AquaLogic Interaction .NET Application Accelerator

Installation and Development Guide

Version 1.0

Document Revised: December 2007

A. Troubleshooting

Welcome

The AquaLogic Interaction .NET Application Accelerator is a collection of libraries and VisualStudio.NET integration features that support easy authoring of ASP.NET 2.0 portlets. Portlets authored using the .NET Application Accelerator can be consumed in both BEA portal environments: AquaLogic Interaction (ALI) and WebLogic Portal (WLP). In AquaLogic Interaction, portlets can be directly consumed using CSP over HTTP. In WebLogic Portal, portlets can be consumed over WSRP.

This guide describes how to install and configure the .NET Application Accelerator, and provides basic instructions on using the .NET Application Accelerator to develop portlets for BEA portal products.

This chapter provides an introduction.

- .NET Application Accelerator Components
- How to Use This Book
- BEA Documentation and Resources

.NET Application Accelerator Components

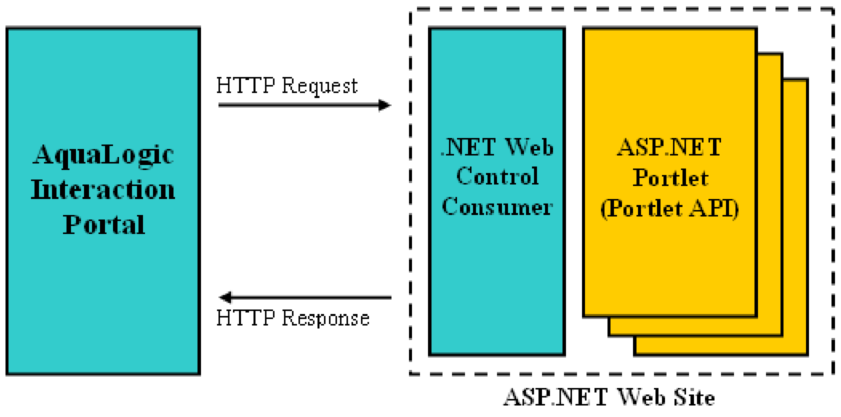
The .NET Application Accelerator makes it easy to author ASP.NET 2.0 portlets in BEA portal environments. The .NET Application Accelerator includes three components:

- **.NET Portlet API 1.0**: The Portlet API is a set of project templates, file templates, and base classes used to create portlets in the VisualStudio.NET environment. The Portlet API includes the IDK 5.3 release and exposes functionality such as preferences and user information from this library in a simplified manner. The Portlet API provides separate project and file templates for ALI and WLP portlets.
  For details on portlet development, see Chapter 4, "Portlet Development - AquaLogic Interaction," and Chapter 5, "Portlet Development - WebLogic Portal."
- **Web Control Consumer (WCC) 3.1**: The Web Control Consumer transforms output rendered by ASP.NET into pages that can be consumed inside a portal.
  For details on configuring and using the WCC, see Chapter 6, "Configuration and Development - .NET Web Control Consumer (WCC)."
  Note: The 3.1 version of the Web Control Consumer supports only ASP.NET 2.0.
- **.NET WSRP Producer 1.0**: The WSRP Producer is a standards-conformant .NET web application that can expose ASP.NET 2.0 applications via the WSRP protocol. The WSRP Producer is required to use the .NET Portlet API in WLP.

The components of the .NET Application Accelerator can be used to build and provide portlets in two configurations that correspond to the two BEA portal offerings, as shown on the next page.

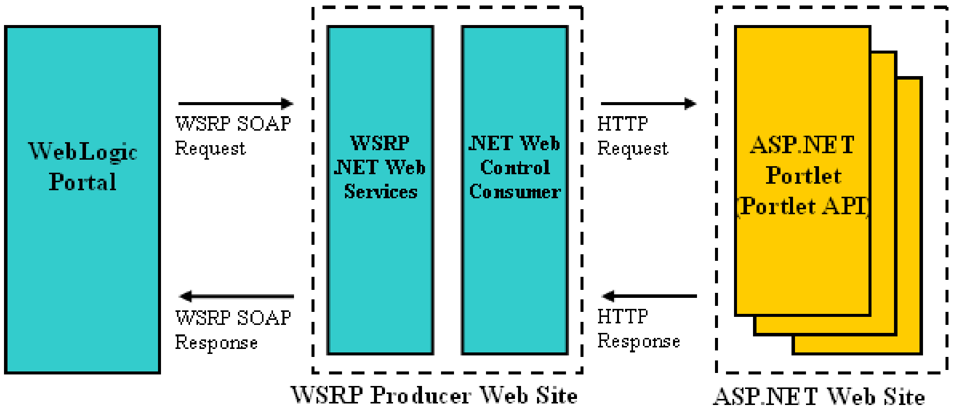
Required Components - AquaLogic Interaction (ALI)

To build portlets for AquaLogic Interaction, the following components are required:


AquaLogic Interaction Portal
HTTP Request
HTTP Response
.NET Web Control Consumer
ASP.NET Portlet (Portlet API)
ASP.NET Web Site

Required Components - WebLogic Portal (WLP)

To build portlets for WebLogic Portal, the following components are required:



How to Use This Book

This guide has been designed to be a quick reference for users with installation and development experience.

Audience

This guide is written for the user responsible for installing and configuring the .NET Application Accelerator. This user must have strong knowledge of the platform operating system, web and application servers, and any other third-party software required for installation.

Organization

This chapter provides information on how to use this guide and describes other resources available to help deploy and implement the .NET Application Accelerator in your portal applications.

This guide also includes the following chapters:

- Chapter 2, “Installation Prerequisites” provides hardware and software requirements, as well as environmental and third-party software prerequisites. You must read this chapter and meet the prerequisites prior to proceeding to the installation.
- Chapter 3, “Installation” provides detailed instructions for installing and configuring the .NET Application Accelerator.
- Chapter 4, “Portlet Development - AquaLogic Interaction” provides information on developing portlets with the .NET Application Accelerator in AquaLogic Interaction.
- Chapter 5, “Portlet Development - WebLogic Portal” provides information on developing portlets with the .NET Application Accelerator in AquaLogic Interaction.
- Chapter 6, “Configuration and Development - .NET Web Control Consumer (WCC)” provides instructions on configuring the .NET WCC and and information on basic development best practices.
- Appendix A, “Troubleshooting” provides instructions for troubleshooting and debugging the .NET Application Accelerator installation.

Typographical Conventions

This book uses the following typographical conventions.

Table 1-1 Typographical Conventions

Convention	Typeface	Example
• File and folder names • Screen elements • Object and method names	**bold**	• Upload **Procedures.doc** to the portal. • Open the **General** folder. • To save your changes, click **Apply Changes**.
• Text you enter • Sample code	`computer`	• Type `Marketing` as the name of your community.

BEA Documentation and Resources

This section describes other documentation and resources provided by BEA.

Table 1-2 BEA Documentation and Resources

Resource	Description
Release Notes	These files are written for .NET Application Accelerator administrators. They include information about new features and known issues in the release.
Developer Guides, Articles, API Documentation, Blogs, Newsgroups, and Sample Code	These resources are provided for developers on the BEA dev2dev site (dev2dev.bea.com).

Table 1-2 BEA Documentation and Resources

Resource	Description
AquaLogic User Interaction Support Center	The AquaLogic User Interaction Support Center is a comprehensive repository for technical information on AquaLogic User Interaction products. From the Support Center, you can access products and documentation, search knowledge base articles, read the latest news and information, participate in a support community, get training, and find tools to meet most of your AquaLogic User Interaction-related needs. The Support Center encompasses the following communities: **Technical Support Center** Submit and track support incidents and feature requests, search the knowledge base, access documentation, and download service packs and hotfixes. **User Group** Visit the User Group section to collaborate with peers and view upcoming meetings. **Product Center** Download products, read Release Notes, access recent product documentation, and view interoperability information. **Developer Center** Download developer tools and documentation, get help with your development project, and interact with other developers via BEA's dev2dev Newsgroups. **Education Services** Find information about available training courses, purchase training credits, and register for upcoming classes. If you do not see the Support Center when you log in to http://support.plumtree.com, contact ALUIsupport@bea.com for the appropriate access privileges.
Technical Support	If you cannot resolve an issue using the above resources, BEA Technical Support is happy to assist. Our staff is available 24 hours a day, 7 days a week to handle all your technical support needs. E-mail: ALUIsupport@bea.com Phone Numbers: U.S.A. +1 866.262.PLUM (7586) or +1 415.263.1696 Europe +44 1494 559127 Australia/NZ +61 2.9923.4030 Asia Pacific +61 2.9931.7822 Singapore +1 800.1811.2027

Installation Prerequisites

Before you run the AquaLogic Interaction .NET Application Accelerator installer, complete the following steps:

1. Read the product release notes for information on compatibility issues, known problems, and workarounds that might affect how you proceed with your deployment.
2. Provision host computers and configure software dependencies for your deployment. For details, see the Hardware and Software Requirements that follow.

Hardware and Software Requirements

The following table summarizes the operating system and software requirements for AquaLogic Interaction .NET Application Accelerator.

Component	Requirement
Host Computer	• Windows Server 2003 SP1 • Microsoft VisualStudio.NET 2005 • Microsoft ASP.NET 2.0 • Microsoft Web Service Enhancements (WSE) 2.0 SP3 • IIS 6.0 (WSRP portlets only)
Portal Platform	• BEA AquaLogic Interaction 6.0 and above • BEA WebLogic Portal 9.2 (with .NET WSRP Producer 1.0)

Installation

This chapter describes how to install and configure the AquaLogic Interaction .NET Application Accelerator. To install .NET Application Accelerator, complete the following steps:

1. Make sure that you have met the prerequisites for installation. For details, see the hardware and software requirements in Chapter 2, "Installation Prerequisites."
2. Install the .NET Application Accelerator components using the instructions that follow.

Installing the AquaLogic Interaction .NET Application Accelerator

Follow the instructions below to install the AquaLogic Interaction .NET Application Accelerator.

1. Copy the installation package to a location on the host server.
2. To launch the installer, click the **BEADotNetApplicationAccelerator.exe** file.
3. Complete the installation wizard pages as described in the table that follows.

Note: To facilitate any troubleshooting that might be required, as well as communication with IT staff and customer support, we recommend keeping the default settings.

Table 3-1 .NET Application Accelerator Installer Wizard Pages for Windows

Wizard Page	Description
License Agreement	Read and accept the license agreement
Choose Components	Select the components to be installed: • **.NET Application Accelerator**: This option installs all the components needed to develop ALI and WSRP (WLP) portlets, including the Portlet API, Web Control Consumer, and WSRP Producer. This includes both the required runtime components and the required image server files. Most developers will typically choose this component. • **Image Service Files for Windows Image Server**: This option installs the URL-addressable resources that must be available for the Web Control Consumer to operate correctly. This component should be selected when these resources must be installed onto a separate static content server such as IIS or Apache. The pages that follow will be based on your selection; some of the pages listed below will not appear.
Choose Install Folder	Enter the path to the folder where the .NET Application Accelerator files should be installed. (This page is shown if the *.NET Application Acclerator* option was selected.)
Select IIS Web Site	Choose an IIS web site option: • **Use Default Web Site**: If you are only developing portlets for ALUI, choose the default option. You can remove the web site from IIS once the installer has completed. • **Use another web site**:If you are developing WSRP portlets for WebLogic Portal, choose this option and configure the site settings on the next page. (This page is shown if the *.NET Application Acclerator* option was selected.)
Specify IIS Web Site Information	Configure the IIS settings for deployment of the WSRP Producer web site. Enter the name of the web site and its unsecured and secured ports. After the installer is complete, the WSRP Producer web site will be available at the following URL: http://<iis-website-address>/wsrpproducer (This page is shown if the *Use another web site* option was selected on the previous page.)

Table 3-1 .NET Application Accelerator Installer Wizard Pages for Windows

Wizard Page	Description
Path to Image Service Files	This page displays the location from which static content is served by the .NET Application Accelerator. If you modify the path, some additional installation steps may be required. (This page is shown if the *Image Service Files for Windows Image Server* option was selected.)
Pre-Installation Summary	Review the installation settings and click **Next** to proceed.
Install Complete	Click **Done** to close the installer.

Once installation is complete, the WSRP Producer web site will be registered with IIS, a set of project and file templates will be available to VisualStudio.NET, and the installation directory will contain necessary resources and libraries.

Portlet Development - AquaLogic Interaction

The.NET Application Accelerator provides ease-of-use features that make it easy to author ALI portlets. All the features available in the IDK are still available, but some of them are much simpler to use, requiring only a single line of code to accomplish tasks that previously took four or five lines. The .NET Application Accelerator has ALI portlet project templates, file templates for portlet and preference editing pages, and a class library that provides easy access to preferences, user properties, profile information, and other portlet information.

Getting Started with ALI Portlet Project and File Templates

The .NET Application Accelerator includes VisualStudio.NET integration for easy authoring of portlets. The provided project templates are pre-configured to include references to required BEA assemblies, including the IDK and new Portlet API libraries. The template is also configured for integration with the ALI PTSpy logging infrastructure and the IDK user profile provider. These settings are available in the site's Web.config file.

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

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

1. Create a new ALI Portlet project in VisualStudio.NET:
   a. Start VisualStudio.NET.
   b. Click **File** | **New** | **Web Site** to create a new project.
   c. Select the **C#** language option.
   d. In the My Templates list, select **ALI Portlet Project**.
   e. Name the project "simpleAliWeb".
2. Create a new ALI portlet page:
   a. In the Solution Explorer, right-click on the root of the project and select **Add New Item…**
   b. In the My Templates list, select **ALI Portlet Page**.

The ALIPortletPage class provides access to functionality described in the sections that follow.

Using Preferences with ALI Portlets

The .NET Application Accelerator provides an easy way to read and write preference values provided to a remote portlet by AquaLogic Interaction. For details on the various levels of preferences, see the AquaLogic User Interaction Development Documentation: http://edocs.bea.com/alui/devdoc/docs60/index.html#Overview_of_the_Portal_Architecture/Plumtree_Settings/PlumtreeDevDoc_Integration_SettingsModel.htm

The .NET Application Accelerator allows portlets to manipulate preferences in two ways. First, **PortletAttributes** corresponding to preference types can be applied to fields in an ALIPortletPage class. The preference types correspond to attributes as follows:

Table 4-1 PortletAttributes

Preference Type	Attribute Name
Admin	AdminPreference
Portlet	PortletPreference
Community	CommunityPreference
CommunityPortlet	CommunityPortletPreference
Session	SessionPreference
User	UserPreference

Preference attributes can be applied to fields with public, protected, or internal visibility with a type of String, int, float, bool, or DateTime. For developers familiar with the IDK, use of a preference attribute on a field in a page replaces calls to read preferences from the IPortletRequest object. The following examples use the UserPreference attribute, but this attribute could be replaced with any of the attributes listed above to read preferences of the corresponding type. Most preferences can be set from a portlet page, but Admin preferences must be set from an ALIPreferencePage (see Creating ALI Preference Pages below).

Fields marked with preference attributes will have their values set after the OnLoad() page event and will have changes persisted during the OnUnload() page event.

The code below reads the "userName" field value from the preference "UserName":

```
[UserPreference(Key = "UserName")]
protected String userName;
```

Note: The *permissions modifier* of the variable you are binding must be either public or protected. Private variables (including those with no modifier) cannot be seen by reflection. The binding will not work and you will not know why. For example, the following code will fail:

```
[UserPreference(Key = "UserName")]
String userName;
```

If you do not include a “Key” attribute, the name of the preference defaults to the name of the variable. The code below reads the “usefulValue” portlet preference.

```
[PortletPreference()]
protected String usefulValue;
```

To set the default value for a preference, use the "DefaultValue" attribute of the preference. The following code sets the "userName" preference to "Anonymous" in the absence of any available user preference value.

```
[UserPreference(Key = "UserName", DefaultValue="Anonymous")]
protected String userName;
```

Once you bind a variable in your C# class to an ALI preference, it is set when the request is first processed, and any changes to the value of the bound variable are persisted in ALI automatically.

If you must read and write dynamic preferences, use the **PreferenceCollection** in the ALIPortletPage base class. To access preferences, use the preference name as shown below:

```
String dynamicPrefValue = this.PortletPreferences["DynamicPrefName"];
this.PortletPreferences["DynamicPrefName"] = dynamicPrefValue;
```

Accessing User Profile Information

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The .NET Application Accelerator Portlet API uses the ProfilerProvider API to expose user profile information sent by AquaLogic Interaction to an ASP.NET web application that exposes portlets. More information about the ProfileProvider model can be found in the MSDN documentation: http://windowssdk.msdn.microsoft.com/en-us/library/014bec1k.aspx

The Web.config file for an ALI portlet project contains a `<profile>` element that defines a Provider to support reading user profile properties from the portal. To use specific profile properties in a page, add the following XML to the `<profile>` definition:

```
<properties>
<add name="Title" type="String" defaultValue="Test" readOnly="true"/>
<add name="Age" type="Int32"/>
</properties>
```

Note: AquaLogic Interaction does not allow user profile properties to be modified in a portlet. If this happens, a warning will be logged and the new property value will be dropped. To avoid suprises, add `readOnly="true"` as an attribute in the `<add...>` node in the properties definition as shown above.

Each property is defined with a name and a type. If you provide a default value, it will be used if the profile property was not found in the current request or if the current request is running outside of the portal gateway.

Profile property values can be read in the page using the following code, which reads the "Title" property from the Profile object in the ALIPortletPage super class:

```
String title = Profile.Title;
```

The name of the profile property must match the name sent by AquaLogic Interaction. For details on user profile properties (also called User Information Settings), see the AquaLogic User Interaction Development Documentation: http://edocs.bea.com/alui/devdoc/docs60/index.html#Overview_of_the_Portal_Architecture/Plumtree_Settings/PlumtreeDevDoc_Integration_SettingsModel.htm

Creating ALI Preference Pages

The .NET Application Accelerator integration with VisualStudio.NET and the Portlet API also provides features to easily create portlet preference pages. For more information on ALI preference pages, see the AquaLogic User Interaction Development Documentation http://edocs.bea.com/alui/devdoc/docs60/index.html#Portlets/Basics/PlumtreeDevDoc_Integration_Portlets_Settings.htm

To create a new ALI preference page, add a new item to your web site using the steps below:

1. In the Solution Explorer, right-click on the root of the project and select **Add New Item…**
2. In the My Templates list, select **ALI Preference Page**.

The ALIPreferencePage template provides the same functionality as an ALIPortletPage and includes additional features for rendering and editing a preference page.

By default, the page renders a title bar with "Finish" and "Cancel" buttons that match the standard ALI preference page style. The appearance and functionality of the title bar can be changed by overriding the `DrawTitleBar()` method from the base class. To complete the preference page, you must implement four methods:

Table 4-2 ALIPreferencePage Methods

Method Name	Description
`GetTitleBarText()`	Read-only property to get the text for the title bar.
`GetFinishButtonText()`	Read-only property to get the text for the finish button.

Table 4-2 ALIPreferencePage Methods

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

When building portlets in non-English locales, be sure to provide internationalized messages in the three methods that return text labels.

Accessing Additional Portlet Information

The .NET Application Accelerator Portlet API also exposes some basic functionality from the Plumtree.Remote.Portlet package. In addition to preferences, a variety of other portlet-specific objects are available from a portlet page that extends the ALIPortletPage.

The IRemoteSession portal session can be accessed using the following code:

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

The user's name can be accessed using the following code:

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

The host page URI can be accessed using the following code:

```
URI hostPage = HostPageURI;
```

More advanced functionality is still available from the IDK classes, which are present in the project's dependencies.

Deploying Portlets in AquaLogic Interaction

To deploy portlets developed with the .NET Application Accelerator in AquaLogic Interaction, follow the normal steps for configuring the associated portal objects: Remote Server, Web Service - Remote Portlet, and Portlet. For detailed information on portlet configuration, see the Developing Portlets section in the AquaLogic User Interaction Development Documentation: http://edocs.bea.com/alui/devdoc/docs60/index.html#Portlets/Basics/PlumtreeDevDoc_Integration_Portlets_Configuration.htm.

Using Logging with the .NET Application Accelerator

ALI Logging Spy is a logging utility that is automatically installed with AquaLogic Interaction or installed as part of the ALI Logging Utilities, a stand-alone package available for download on dev2dev. ALI Logging Spy listens for log messages transmitted using network broadcast. By configuring a list of message senders, you can listen in on log traffic from multiple BEA applications, live, in a single console.

To add a message listener to your Spy installation, run Spy, select **View | Set Filters**. In the Filter Settings dialog, click **Edit | Add Message Sender** and choose a message sender.

The .NET Application Accelerator has three logging message senders:

- **.NET WSRP Producer**: This is the stand-alone web application designed to serve as a WSRP producer for WLP. It surfaces ASP.NET web applications in WLP by transforming WSRP SOAP requests into HTTP requests, and transforming content so it works in WLP. This name is not configurable and transmits all messages related to the WSRP Producer.
- **ALI .NET Application Accelerator**: Name reserved for future use. No logging traffic is sent by this sender. The WSRP producer web application also has a "loggingName" entry in web.config that is not currently used.
- **WSRPPortletProjectXX**: This name is defined in the web.config file of the ASP.NET application built using the .NET Portlet API (the templates installed in Visual Studio .NET 2005). The following line in web.config configures this logging application name:
  ```
  <add key="ptedk.LoggingApplicationName" value="WSRPPortletProjectXX"/>
  ```
  This message sender transmits log messages from the Web Control Consumer and the .NET Portlet API, as well as the IDK.

For more information on configuring ALI Logging Spy, see the ALI Logging Utilities section of the AquaLogic User Interaction Development Documentation:
http://edocs.bea.com/alui/devdoc/docs60/index.html#Plumtree_Development_Environment/Debugging/PlumtreeDevDoc_DevEnvEDK_Debugging_Intro.htm

Portlet Development - WebLogic Portal

WSRP (Web Services for Remote Portlets) is a W3C standard for consuming one or more remote markup "Producers" from a markup "Consumer". The .NET Application Accelerator provides a WSRP Producer for ASP.NET that can be used to produce portlets for the WebLogic Portal's WSRP Consumer.

Note: In order to produce portlets, the WSRP Producer must be installed and correctly configured with the local IIS instance. A working WSRP Producer provides a WSRP WSDL available at the following URL: http://<iis-website-address>/wsrpproducer/1.0/WSRPService.wsdl

This URL will be required below when referencing a .NET WSRP Producer from the WebLogic Portal WSRP Consumer. To ensure that the WSRP producer is running, paste this URL into the address bar of a web browser.

The .NET Application Accelerator includes VisualStudio.NET integration for easy authoring of WSRP portlets. The provided WSRP web site project template is pre-configured to include references to required BEA assemblies, including the new Portlet API libraries. These settings are available in the site's Web.config file.

A new WSRP Portlet project also includes a simple WSRP portlet page in the Default.aspx / Default.aspx.cs files. This page extends the WSRPPortletPage base class, which provides access to preference and user information.

Note: ALI Portlet and Preference Page templates should not be used for WSRP portlets.

This chapter includes the following sections:

- Getting Started with WSRP Portlet Project and File Templates
- Using WSRP Preferences with WSRP Portlets
- Using User Profile Preferences with WSRP Portlets
- Registering a Portlet with the WSRP Producer
- Consuming a .NET WSRP Portlet from WebLogic Portal
- Deploying the WSRP Producer
- Retrieving SAML Tokens from WLP

Getting Started with WSRP Portlet Project and File Templates

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

1. Create a new WSRP Portlet project in VisualStudio.NET:
   a. Start VisualStudio.NET.
   b. Click **File** | **New** | **Web Site** to create a new project.
   c. Select the **C#** language option.
   d. In the My Templates list, select **WSRP Portlet Project**.
   e. Name the project "firstWsrpWeb".
2. Create a new WSRP portlet page:
   a. In the Solution Explorer, right-click on the root of the project and select **Add New Item…**
   b. In the My Templates list, select **WSRP Portlet Page**.
   c. Save the file as “HelloWorld.aspx”.

The WSRPPortletPage class provides access to functionality described in the sections that follow.

Using WSRP Preferences with WSRP Portlets

The .NET WSRP Producer supports user scoped preferences that can be declaratively created in a WSRPPortletPage instance and are managed by the WSRPPortletPage itself. For example, a WSRP preference can be declared simply using an attribute as shown in the code below:

```
[PortletStateValue()]
protected String colorSelection;
```

This will extract a preference named colorSelection from the PortletRequest and set the value in this field. This is supported for fields declared as public, protected, and internal and are of type String, int, float, bool, or DateTime. Preference values will be available after the OnLoad() page event. Any changes to the value of a field marked with the PortletStateValue attribute after the OnLoad() event will be set as a new preference value during the OnUnload() event.

In the absence of the value for a preference in the PortletRequest, the value of the preference will default to the intrinsic default value of the preference type in the language. To provide a default value, use the the attribute's "defaultValue" field. For example:

```
[PortletStateValue(DefaultValue="orange")]
protected String colorSelection;
```

To decouple the name of the field in the page class and the name of the preference, set the attribute's "Key" attribute. For example,

```
[PortletStateValue(Key="colorlSelection", DefaultValue="orange")]
protected String theColor;
```

This will extract the preference named "colorSelection" and apply it to the field "theColor".

For a full description of the API, see the complete class documentation for the WSRP part of the Portlet API, which is installed into the VisualStudio.NET 2005 Integrated Help System. Standard NDoc HTML files are also installed in: $PT_HOME/portletapi/1.0/Documentation

Note: WebLogic Portal supports WSRP Preferences, but in order to use preferences, the .portal file describing a portal in WebLogic Portal must be converted from "file" mode to "streaming" mode. For details, see "Creating a Desktop" on the following page: http://edocs.bea.com/wlp/docs92/portals/creating_staging.html

To get started with WSRP portlet preferences in ASP.NET, create a new WSRP portlet page with these steps:

1. Create a new portlet using the WSRPPortletPage template:
   a. In Solution Explorer, right-click on the root of the web site project and select **Add New Item…**

   b. In the **My Templates** list, select **WSRP Portlet Page**.

   c. Name the page "SimplePreference.aspx".

2. Add a preference to the portlet by adding this code to the code-behind file for SimplePreference.aspx: [TODO: code sample. PM.]

3. Register the new SimplePreference portlet with the WSRP Producer as described in Registering a Portlet with the WSRP Producer.

4. Follow the steps to create a new portlet in WebLogic Portal and add the new portlet to the existing streamed portal. In order to make changes to a streamed portal, the new SimplePreference portlet must be added to the portal via the online administration tools for the portal application. Also, in order to modify preferences, a user must be logged into WebLogic Portal. See the WebLogic Portal documentation for additional information for configuring login.

5. Log in to the portal and access the location of the portal in a web browser. Refreshing the page will cause the preference value to change and keep track of the number of times the portlet has been accessed by the user.

Note: In WSRP portlets, a change in preference state is not reflected until the next request to a WSRP portlet. This is because transmission of the preference and persistence of the new preference value happens in the response at the end of the preference editing request. The new value for the preference will be available in the next portlet request.

Using User Profile Preferences with WSRP Portlets

ASP.NET 2.0 introduced the Provider model as a design pattern for plugging data providers into the framework. The Portlet API makes use of the ProfileProvider to expose user profile information sent by WebLogic Portal over WSRP to an ASP.NET web application. More information about the ProfileProvider model can be found in the MSDN documentation here: http://windowssdk.msdn.microsoft.com/en-us/library/014bec1k.aspx

The Web.config file for a .NET WSRP Portlet project contains a `<profile>` element that defines a Provider to supporting accessing user profile properties. In order for user profile properties from WebLogic Portal to flow over WSRP into the .NET web application, they must be registered in the Web.config file as follows:

```
<properties>
```

```
<add name="CustomPropertiesTitle" type="string" defaultValue="Test
Title"/>
<add name="MyProfileAge" type="Int32" />
<add name="MyProfileBirthday" type="DateTime" />

</properties>
```

Each property must have a "name" and a "type" attribute and can include an optional "defaultValue" attribute. These values can be accessed in a WSRPPortletPage by using the Profile property of the superclass as shown in the code below:

```
int age = Profile.MyProfileAge;
```

Note: Property names must follow a specific naming format to match the user profile property naming conventions used by WebLogic Portal. User profiles consist of a named property set of named properties. The rule for naming a property in a user profile is to use the capitalized name of the property set concatenated with the capitalized name of the property. In the example above, the property set is named "MyProfile" and has two properties "Age" and "Birthday". For more information, see the WLP documentation: http://e-docs.bea.com/wlp/docs92/users/developuserprofiles.html

The .NET WSRP Producer must also be configured to send these properties from the WSRP message to the ASP.NET web application. This requires adding a `<user-profile>` element to a `<portlet>` and configuring the property names that should be passed through the WSRP Producer. For example, add the following code to the end of a `<portlet>` element to pass the properties described above:

```
<user-profile>

  <item>CustomProperties/Title</item>

  <item>MyProfile/*</item>

</user-profile>
```

The properties from a property set can be filtered individually using a string with the format `<propertyset-name>/<propertyname>`. Using "*" for the `<propertyname>` will pass all properties from a property set from the WSRP Producer to the ASP.NET web application.

Registering a Portlet with the WSRP Producer

Once the portlet content has been created, the URLs of the portlet content and additional metadata must be configured in the WSRP Producer. All the portlets available from the WSRP Producer must be registered in the /portlets.xml file at the root of the WSRP Producer's web site.

To add a portlet, edit the **portlets.xml** file in a text editor and add the following XML between the enclosing `<portlets>` elements:

```
<?xml version="1.0" ?>
<portlets xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- Configure the HelloWorld portlet -->
<portlet>
  <handle>HelloWorldPortlet</handle>
  <portlet-name>Hello World Portlet</portlet-name>
  <view_url>/firstWsrpWeb/HelloWorld.aspx</view_url>
  <expiration-cache>10</expiration-cache>
  <supported-locale>en</supported-locale>
  <description lang="en">HelloWorld WSRP Portlet</description>
  <display-name lang="en">HelloWorld Portlet</display-name>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>View</portlet-mode>
  </supports>
  <portlet-info>
    <title>Hello World Portlet</title>
    <short-title>View</short-title>
  </portlet-info>
</portlet>
</portlets>
```

This configures the URL for the portlet content in the `<view_url>` element and adds additional information about the default title, supported view mode and mime types, and provides the portlet a WSRP handle name. Once this step is complete, the portlet can be consumed by WLP.

Consuming a .NET WSRP Portlet from WebLogic Portal

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

In order to create a Remote Portlet produced by the .NET WSRP Producer, you must know the address of the Producer's WSRPService.wsdl file. The URL will vary depending on how IIS is configured to host the WSRP Producer, but it should be similar to the following: http://192.168.1.1:1234/wsrpproducer/1.0/WSRPService.wsdl

To create the Remote Portlet using the WebLogic Portal IDE tools, follow the steps below:

1. Click **File** | **New** | **Portlet**.
2. Under **Portlet Type**, select **Remote Portlet**.
3. Select the Producer from the **Select Producer** menu. If this is the first time the WSRP Producer is used to provide remote portlets to WebLogic Portal, follow the steps below:
   a. Select **Find Producer** and enter the URL to the WSRPService.wsdl file as `http://<server>:<port>/wsrpproducer/1.0/WSRPService.wsdl`
   b. Click **Retrieve** to obtain the list of available remote portlets from the .NET WSRP Producer. Click **Next**.
   c. The page shows the list of available, remote portlets listed in the WSRP Producer's portlets.xml file. Select one and click **Next**.
   d. In the **Producer's Handle** field, enter a name that is descriptive of the .NET WSRP Producer. This name will be displayed in the Select Producer menu.
4. Create the portlet and add any necessary functionality.
5. Once the .portlet file is created, add it to the .portal file. (For instructions on creating a .portal file, see http://e-docs.bea.com/wlp/docs92/portals/develop_portals.html#wp1011957.)

For detailed information on creating remote portlets, see http://edocs.bea.com/wlp/docs92/federation/Chap-Build_Simple_Remote.html#wp1016477

The .portal file containing the ASP.NET portlet can be rendered by rebuilding and redeploying the Portal application and accessing the URL of the .portal file. Consult the WebLogic Portal documentation for information on creating and deploying portal and portlet resources.

If problems arise following these steps, see Appendix A, "Troubleshooting.".

Deploying the WSRP Producer

This section is applicable only to the WSRP Producer. During installation, the WSRP Producer web site is deployed to IIS based on the configuration settings provided to the installer. This configuration information is used to parameterize the WSRP Producer's WSDL file which describes the locations of various WSRP web services. After installation has completed, any changes to the IP address, port number, or web site deployment path must be made to this file: **$DNAA_HOME/wsrpproducer/1.0/webapp/wsrpproducer/1.0/WSRPService.wsdl**

Forgetting to make these changes will result in failure of the WSRP Producer to locate required WSRP web services. To make changes, simply edit WSRPService.wsdl in a text editor and change the necessary entries. For example, to update the location of the WSRP Producer, change the following entry:

```
<soap:address
location="http://192.168.1.1:1234/wsrpproducer/1.0/WSRPBaseService.asmx
"/>
```

```
to the correct IP address and port, for example:
```

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

Additional configuration of the WSRP Producer web site can be made via the IIS administrative tools.

Retrieving SAML Tokens from WLP

SAML assertions are sent from WLP over WSRP when WLP is configured to send them. SAML configuration are often unique; to learn more about SAML configuration, read *Establishing WSRP Security with SAML* (http://e-docs.bea.com/wlp/docs92/federation/Chap-Security-SAML.html). You may also want to read *Configuring Single Sign-On with Web Browsers and HTTP Clients* (http://e-docs.bea.com/wls/docs92/secmanage/saml.html).

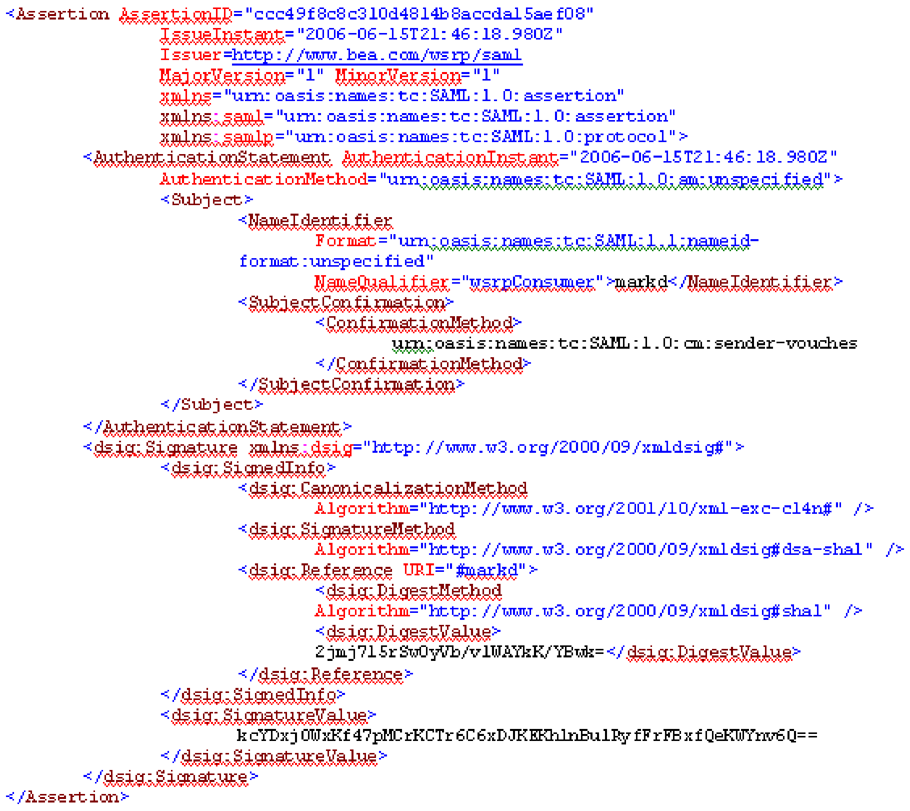
The SAML token is passed directly to the .NET Portlet API as part of the `User` property of the `WSRPBasePage`, the base class for `WSRPPortletPage`. You can access the SAML token using the `SAMLToken` field in `WSRPPortletUser`.

The following code accesses the `SAMLToken` field from `WSRPPortletPage`:

```
XmlElement samlTok = User.SAMLToken;
```

If no SAML token was sent, this property is set to null.

Below is an example of a SAML token returned from `WSRPPortletUser.SAMLToken`:



```
<Assertion AssertionID="ccc49f8c8c310d4814b8accda15aef08"
        IssueInstant="2006-06-15T21:46:18.980Z"
        Issuer=http://www.bea.com/wsrp/saml
        MajorVersion="1" MinorVersion="1"
        xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
    <AuthenticationStatement AuthenticationInstant="2006-06-15T21:46:18.980Z"
        AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:unspecified">
        <Subject>
            <NameIdentifier
                Format="urn:oasis:names:tc:SAML:1.1:nameid-
            format:unspecified"
                NameQualifier="wsrpConsumer">markd</NameIdentifier>
            <SubjectConfirmation>
                <ConfirmationMethod>
                    urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
                </ConfirmationMethod>
            </SubjectConfirmation>
        </Subject>
    </AuthenticationStatement>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <dsig:SignedInfo>
            <dsig:CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            <dsig:SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
            <dsig:Reference URI="#markd">
                <dsig:DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <dsig:DigestValue>
                2jmj7l5rSw0yVb/vlWAYkK/YBwk=</dsig:DigestValue>
            </dsig:Reference>
        </dsig:SignedInfo>
        <dsig:SignatureValue>
            kcYDxj0WxKf47pMCrKCTr6C6xDJKEKhlnBulRyfFrFBxfQeKWYnv6Q==
        </dsig:SignatureValue>
    </dsig:Signature>
</Assertion>
```

Configuration and Development - .NET Web Control Consumer (WCC)

The .NET Application Accelerator's Web Control Consumer (WCC) package includes support for standard .NET Web Controls within AquaLogic Interaction. This chapter assumes that the reader is familiar with standard portlet development practices as detailed in the AquaLogic Interaction Development Documentation (http://edocs.bea.com/alui/devdoc/docs60/index.html).

Introduction - Why WCC?

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

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

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

Additional Configuration Steps

This section explains additional configuration steps for the .NET Web Control Consumer.

Alternative Image Service Access: If the URL used to access your Image Service internally is different from the URL used for external access, and your portlet server is hosted internally, follow these steps to define the Image Service address.

Logging: AquaLogic Interaction Logging Spy logging functionality is enabled by default. These instructions explain how to disable logging by either ALI Logging Spy or log4net.

Future Framework Support: Use these instructions to add support for any .NET Framework version that uses JavaScript WebUIValidation.

Portlet Configuration: To configure an existing .NET application to be used in AquaLogic Interaction, use these steps to make a new entry in the Web application's web.config file.

Alternative Image Service Access

Some network configurations require that the portal Image Service (previously called Image Server) is accessed through a different URL internally and externally. This can be a problem for portlet servers that are hosted internally because AquaLogic Interaction always sends the external Image Service URL (the portlet must contact the Image Service to retrieve auxiliary JavaScript files).

If this is the case, set up a mapping to allow the portlet server to determine the internal Image Service address. This can be achieved in two ways.

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

1. Open imageserver.mapping.xml in a text editor. The file is located in the .NET Web Controls installation folder, under \settings\config.
2. Add a new entry for the internal Image Service, as shown below:

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

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

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

   Both mappings do the same thing in this case. For details on syntax, see .NET regular expressions.
3. Save the imageserver.mapping.xml file.
4. Restart IIS or re-save the config.xml file located in the same directory (this instructs the Web Controls to reload all configuration settings).

The second option is to set an alternative Image Service URL to override the external URL, using an Administrative preference. The default setting name is PTWC.Mapping.Override (this name can be changed in the HttpPipe.xml file). You must create an administrative preferences page to set the preference.

Note: This granularity of configuration is not generally necessary except for specific remote portlets that access the Image Service through a specific unique URL.

Logging

The .NET Web Control Consumer supports logging, which is primarily used for troubleshooting, either in the case of an internal error or configuration problem, or to view operations that the .NET Web Control Consumer is performing. By default, AquaLogic Interaction Logging Spy and one file logger logging at INFO level are configured.

The file logger is a log4net rolling log file appender, which logs to the settings/logs directory in the installation directory. Any problems encountered by the .NET Web Control Consumer are logged here. Under normal circumstances, you will not need to view this log unless problems occur.

When configuring additional options (such as internal Image Service mapping), you can confirm your changes by running AquaLogic Interaction Logging Spy (ptwc/3.0/bin/assemblies/ptspy.exe) and configuring the "Web Services - Portlet API" component to run at DEBUG level and making sure that the options are correctly applied.

To disable existing logging, remove its `<logconfig>` section from the config.xml file.

Future Framework Support

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

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

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

Note: Only WebUIValidation version 125 is currently supported.

Portlet Configuration

To configure an existing .NET Web application to be used in AquaLogic Interaction, you must make a new entry in the Web application's web.config file. In the /configuration/system.web node, add a new node that specifies the HttpModule class and containing assembly, as shown:

```
<httpModules>

   <add type="Com.Plumtree.Remote.Loader.TransformerProxy,
   Aqualogic.WCLoader, Version=3.0.0.0, Culture=neutral,
   PublicKeyToken=d0e882dd51ca12c5" name="PTWCFilterHttpModule" />

</httpModules>
```

Note: The <add> section must be entered on one line. If you cut and paste the code from this document, make sure to remove the word wrap breaks or your code will not work.

Once you add this entry, all gatewayed requests to the Web application are transformed for use with AquaLogic Interaction. You can still access your Web application directly; the module is bypassed for any request that does not originate from a portal server. For example templates, see the following folders in the installation directory: \deploy\web.config (sample web.config file) and \deploy\web.config.node (node only).

Note: Any .NET portlet page that performs postbacks (including auto-postback) must be gatewayed (which is included in the list of gateway prefixes in the Portlet Editor). For details on configuring the gateway, see the AquaLogic Interaction Development Documentation or the portal online help.

Your IIS Web Application Server must have direct Web access to the Plumtree Image Service.

Development Best Practices

This section includes additional best practices related to the .NET Web Control Consumer.

- Postbacks
- Unique Naming
- Submit Buttons
- File Upload
- Custom JavaScript
- Filter Disabling
- Development Checklist

For more tips and tricks, see the .NET WCC FAQ: http://edocs.bea.com/alui/devdoc/docs60/index.html#References/FAQ_PlumtreeNETWebControls.html.

Postbacks

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

- Because all POSTs are handled by JavaScript, they do not become part of the browser history. This means that JavaScript functions will not work; the Back and Forward buttons of the browser will skip any clicks made on Web Controls that are handled in this way.
- All requests from a portal server are transformed; navigating to the portlet directly does not yield the same HTML that is returned to the portal server. This allows the ASPX page to be accessed directly if required (the modified page would not function outside the portal environment). To examine the HTML going to the portal server, use a trace tool between the portal and the portlet server to intercept the HTML.
- All client-side validation script is transformed, but since ASP.NET only generates client-side script for specific versions of Internet Explorer (IE), nothing is altered for browsers other than IE.
- All .NET portlets are wrapped in an HTML span. Do not assume that the portlet is at any specific point in the HTML structure of the page; portlet parent elements are usually HTML table cells, but not in this case.
- The `_doPostBack` method is removed (a similar function is included in an additional JavaScript file). Beware of hooking into this .NET-provided function directly; both the function name and argument list are modified. Any direct calls to this function should also be modified automatically, however any non-standard calling conventions could potentially not be recognized by the module. To programmatically perform a postback from JavaScript, make the call `_doPostBack('','');`.
- Always use FlowLayout because GridLayout uses absolute positioning.
- The names for HTML spans (for validators) and forms (required for all pages that perform a postback) are appended with a unique ID (portlet ID), to avoid naming collisions on a portal page. Do not refer to these names directly in JavaScript. (If it is absolutely required, you can assume that they will be named <original-name>_<portlet-id>.)

Unique Naming

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

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

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

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

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

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

Submit Buttons

Submit buttons will normally be modified to remain in the portal context. This was not the case in the 2.1.x release; in that release, they required an additional ptrender attribute. This now defaults to true, although it can be disabled by adding a `ptrender=false` attribute to the button.

Note: As in the 2.0.x release, it is still possible to place a ptrender tag on the form itself. This means that anything attempting to submit the form will cause it to be posted back in-page. However, we recommend using the tag on the buttons, unless it is specifically required (for example, a third-party control is attempting to submit the form directly).

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

File Upload

In-page uploading is not supported. Any upload forms (multipart/form-data) will not be posting in-page and will post as normal through the gateway. Make sure to redirect back to the hosting page if uploading is required, or to perform the upload in a popup or non-aggregated page.

Custom JavaScript

Because each page refresh dynamically writes the HTML to re-render the portlet, custom JavaScript can pose a problem. Because any JavaScript is now executing as the portlet is rendering, there are some functions that will not work correctly. Any document.write(..) calls will not work.

To call a script each time the page is refreshed (or every time the portlet is re-rendered), use the **`rerenderPortletID PCC`** event. This event is available only to .NET portlets and is named using the current portlet ID.

For example, to create a JavaScript alert each time the portlet is rendered, use the code below.

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

```
<pt:namespace pt:token="$$PORTLET_ID$$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>

function alertMe_$$PORTLET_ID$$, () {

      alert("Portlet Rendering!");

}

alertMe_$$PORTLET_ID$$;//call the first time the portlet is loaded
```

Then register for the event in the code behind:

```
private void Page_Load(object sender, System.EventArgs e)

{

if(!IsPostBack)

RegisterClientScriptBlock("rerender", "<script
language=\"javascript\">document.PCC.RegisterForWindowEvent('rerender.$
$PORTLET_ID$$', alertMe_$$PORTLET_ID$$)</script>");
```

In the example above, $$PORTLET_ID$$ is replaced by the portlet ID using the pt:token adaptive tag. You can also use the IDK to extract the portlet ID from the request. See the AquaLogic Interaction Developer Documentation for more information on adaptive tags.

Filter Disabling

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

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

This will disable filtering for the current request only.

Development Checklist

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

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

Migrating Existing WCC Applications

This page provides basic instructions for migrating applications built with the 2.1.x version of the WCC. Migrating from versions 2.2.x and above requires no changes to the existing code.

2.1.x to 3.0

Even though 3.0 and 2.1.x are very similar, there are some differences that might require some changes to existing code.

JavaScript Execution

In 2.1.x and earlier, JavaScript was executed only upon the first rendering of the page. Some applications may be using this assumption that inline code will be run only once, therefore, as this is no longer true, it must be modified to run only once. This can be done easily on the server, by including the code only if IsPostBack == false. This can also be done on the client a few ways, such as:

```
<script>
if(!already_run)
{
//run my code here...
var already_run = true;
}
</script>
```

Many existing applications register for PCC events upon the initial render; this code must be changed to prevent multiple PCC registrations.

ptrender for Buttons

In 2.1.x, buttons would NOT submit a form in-page by default; adding a ptrender=true attribute to the button would cause this behavior. The default behavior now is to submit the form in-page, which can be disabled by added a ptrender=false attribute to the button. Therefore if there are buttons relying on the default behavior of not performing an in-page refresh, these must be changed to include a ptrender=false attribute (note that including a ptrender=true attribute now has no effect).

Troubleshooting

This chapter includes additional information on troubleshooting WSRP portlets.

- Debugging the WSRP Producer Deployment
- Preferences Don't Work Over WSRP in WebLogic Portal
- Using the WebLogic Portal SOAP Monitor to Debug WSRP Traffic
- Images Do Not Appear in WSRP Portlets
- Portlet Styles and Themes are Not Displayed

Debugging the WSRP Producer Deployment

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

Preferences Don't Work Over WSRP in WebLogic Portal

Ensure that the WebLogic .portal file has been converted to streaming mode using the instructions in the WebLogic Portal documentation:

http://edocs.bea.com/wlp/docs92/portals/intro_to_portal_dev.html#wp1002092
http://edocs.bea.com/wlp/docs92/portals/vistools_config.html#wp1027919

Using the WebLogic Portal SOAP Monitor to Debug WSRP Traffic

WebLogic Portal provides a SOAP monitor to keep track of the SOAP traffic between a WSRP Consumer and Producer. This can help when debugging problems between these two. Instructions for accessing the SOAP monitor can be found here:

http://e-docs.bea.com/wlp/docs92/federation/Chap-Best_Practices.html#wp1010882

When using the SOAP monitor, it is often helpful to create a new portal file with a single portlet which makes simpler to track the traffic between the Consumer and a single Producer.

Images Do Not Appear in WSRP Portlets

The WebLogic Portal WSRP Consumer allows access to resources hosted in remote web applications acting as WSRP Producers. The AquaLogic Interaction .NET Application Accelerator's WSRP Producer is a web application that is separate from the ASP.NET web sites being consumed. As such, WebLogic Portal will need to be configured to allow URL addressable content such as images, style sheets, and JavaScript includes to be returned to WebLogic Portal and a user's web browser.

To do this, a `ResourceConnectionFilter` must be added to the WebLogic Portal applications as shown below.

Note: This code sample works with WebLogic Portal 9.2; for instructions for 8.1.x, see the link under References below.

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

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

```
<!--  WLP 9.2 ResourceProxyServlet -->
    <servlet>
<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</serv
let-name>
<servlet-class>com.bea.wsrp.consumer.resource.ResourceProxyServlet</ser
vlet-class>
        <init-param>
            <param-name>resourceConnectionFilter</param-name>
<param-value>local.AllowAllResourceConnectionFilter</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
<servlet-name>com.bea.wsrp.consumer.resource.ResourceProxyServlet</serv
let-name>
        <url-pattern>/resource/*</url-pattern>
    </servlet-mapping>
```

Note: This ResourceConnectionFilter is an example and is not suitable for production. In production environments, we recommend that applications constrain allowable URLs to those that are known to produce remote ASP.NET web sites.

References

WLP 9.2:
http://edocs.bea.com/wlp/docs92/javadoc/com/bea/wsrp/consumer/resource/ResourceConnectionFilter.html

WLP 8.1:
http://e-docs.bea.com/wlp/docs81/wsrp/security.html

Portlet Styles and Themes are Not Displayed

Some portals strictly enforce the HTML 4.01 DTD (http://www.w3.org/TR/html4/strict.dtd). Portlets that use deprecated HTML might not be displayed correctly. Confirm the DTD used by the target portal and design portlets accordingly.