# Oracle® Fusion Middleware

## Developer's Guide for Oracle WebCenter Ensemble

10*g* Release 3 (10.3.0.1.0)

**E14115-02**

July 2009

Provides instructions for setting up a development environment and developing services for Oracle WebCenter Ensemble.

Oracle Fusion Middleware Developer's Guide for Oracle WebCenter Ensemble, 10*g* Release 3 (10.3.0.1.0)

E14115-02

# Contents

## 2 Oracle WebCenter Ensemble Pagelet Development

## 3 Oracle WebCenter Ensemble Login Customization

## 4 Oracle WebCenter Ensemble REST APIs

## 5 Oracle WebCenter Ensemble API Libraries

## 6 Additional Development References

# Preface

This guide provides instructions for setting up a development environment and developing services for Oracle WebCenter Ensemble.

## Audience

This document is intended for software developers responsible for creating external applications that need to utilize Oracle WebCenter Ensemble. The audience of this documentation is assumed to be proficient in developing applications that use SOAP web services.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

### Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

# Related Documents

For more information, see the following documents in the Oracle WebCenter Ensemble 10*g* Release 3 (10.3.0.1.0) documentation set:

- *Oracle WebCenter Ensemble Release Notes*
- *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle WebCenter Ensemble*
- *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble*
- *Oracle Fusion Middleware Web Service Developer's Guide for Oracle WebCenter Interaction*

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

# 1

# Oracle WebCenter Ensemble Development Environment

If you are developing services for Oracle WebCenter Ensemble, you will need to understand the system and prepare your development environment for use with the Oracle WebCenter Interaction Development Kit (IDK).

This chapter contains instructions for setting up an Oracle WebCenter Interaction Development Kit (IDK) development environment, and important background information on the Oracle WebCenter Ensemble development environment.

- Section 1.1, "Oracle WebCenter Interaction Development Kit (IDK) Projects": This section provides step-by-step instructions for the most common tasks in setting up a development environment.
- Section 1.2, "Oracle WebCenter Interaction Logging Utilities": Oracle WebCenter Interaction Logging Utilities are a collection of debugging and logging solutions available for use in Oracle WebCenter Ensemble.
- Section 1.3, "About Server Communication and the Proxy": This section explains how Oracle WebCenter Ensemble acts as a proxy server, brokering transactions between client computers and external resources. This section also provides detailed information on HTTP and CSP, the protocols that define the syntax of communication between Oracle WebCenter Ensemble and external resources.

## 1.1 Oracle WebCenter Interaction Development Kit (IDK) Projects

The following sections provide step-by-step instructions for the most common tasks in setting up a development environment. For details on installing or downloading the Oracle WebCenter Interaction Development Kit (IDK), see the installation guide on Oracle Technology Network at `http://www.oracle.com/technology/index.html`.

**Java**

- Section 1.1.1, "Java: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Eclipse"
- Section 1.1.2, "Java: Deploying a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Eclipse"
- Section 1.1.3, "Java: Debugging a Custom Oracle WebCenter Interaction Development Kit (IDK) Project"

**.NET**

- Section 1.1.4, ".NET: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Visual Studio"
- Section 1.1.5, ".NET: Deploying a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in IIS"

## 1.1.1 Java: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Eclipse

This section describes how to set up a custom Java Oracle WebCenter Interaction Development Kit (IDK) project in Eclipse. The process is different depending on whether or not Eclipse Web Tools Platform (WTP) is installed:

- Section 1.1.1.1, "Eclipse Stand-Alone (without WTP)"
- Section 1.1.1.2, "Eclipse with WTP"

**Note:** These instructions assume you have installed the **Java** version of the Oracle WebCenter Interaction Development Kit (IDK).

### 1.1.1.1 Eclipse Stand-Alone (without WTP)

These instructions describe how to set up a custom Java Oracle WebCenter Interaction Development Kit (IDK) project in Eclipse stand-alone, without Web Tools Platform (WTP) installed.

1. Open Eclipse and click **File** > **New** > **Project**.
2. Type the **Project Name** (for example, "idkproject"). Click **Next** and **Finish**.
3. In the Package Explorer in Eclipse, right-click on the new project and click **Properties** > **Java Build Path** > **Libraries** > **Add External Jars**.
4. Select the *.jar files from the IDK installation directory under the idk\<version>\devkit\java\WEB-INF\lib directory. Click **OK**.

### 1.1.1.2 Eclipse with WTP

These steps describe how to set up a custom Java Oracle WebCenter Interaction Development Kit (IDK) project in Eclipse with Web Tools Platform (WTP) installed.

1. Open Eclipse and click **File** > **New** > **Other** > **Web** > **Dynamic Web Project**.
2. Type the **Project Name** (for example, "idkproject").
3. Choose a **Target Runtime** from the drop-down list. If you have not previously configured a server runtime, click **New** to configure your Apache Tomcat setup.
4. Click **Finish** to complete the Dynamic Web Project wizard.
5. Import the IDK Web project template:
   a. Right-click the project in the Project Explorer and click **Import** > **General** > **File System**.
   b. To define the **From directory** field, navigate to the IDK root directory and select the **\devkit\WEB-INF** folder.
   c. Change the **Into folder** field to **<project name>/WebContent/WEB-INF**.
   d. Click **Finish**.

---

**Note:** The Eclipse Web project view hides the imported JARs stored in WEB-INF/lib and puts those files under ./Java Resources/src/Libraries/Web App Libraries.

---

## 1.1.2 Java: Deploying a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Eclipse

These steps describe how to deploy a custom Java Oracle WebCenter Interaction Development Kit (IDK) project in Eclipse. The process is different depending on whether or not Web Tools Platform (WTP) is installed:

- Section 1.1.2.1, "Eclipse Stand-Alone (without WTP)"
- Section 1.1.2.2, "Eclipse with WTP"

---

**Note:** The instructions below are for Apache Tomcat or Oracle WebLogic. For IBM WebSphere, you must create a .war or .ear file that is compatible with IBM WebSphere. You must first create an appropriate server-config.wsdd using the Oracle WebCenter Interaction Development Kit (IDK) DeployServlet or the supplied service wsdd files. See the IBM WebSphere documentation for detailed instructions.

---

### 1.1.2.1 Eclipse Stand-Alone (without WTP)

These steps describe how to deploy a custom Java IDK project in Eclipse stand-alone (without Web Tools Platform (WTP) installed).

1. Deploy the Oracle WebCenter Interaction Development Kit (IDK) in your application server:
    a. Create a folder for the custom project in the application server's **\webapps** directory. (For example, if Apache Tomcat is installed in C:\tomcat and the project name is "idkproject", the path would be C:\tomcat\webapps\idkproject.)
    b. Navigate to the IDK installation directory and copy the **WEB-INF** and its **\LIB** subfolder to the directory you created in the previous step. This loads Apache AXIS into the application server.
    c. Confirm that Apache AXIS is available by opening the following page in a browser: http://<hostname:port>/<projectname>/servlet/AxisServlet. (Change <hostname:port> to fit your application server, for example, localhost:8080 for Apache Tomcat. Change <projectname> to the name of the folder you created in step 1a.) The browser should display the message "And now... Some Services" and a list of installed services.
2. Compile the class that implements the IDK interface(s) and copy the entire package structure to the appropriate location in your web application, usually the \WEB-INF\classes directory.
3. Content services, identity services and SCI pages require additional configuration. You must add the custom class to the appropriate *Impl keys in the web.xml file in the WEB-INF directory. For details, see XXX_missing x-ref to ref_idk_deploymentimplkeys.dita_XXX.
4. Start your application server. In most cases, you must restart your application server after copying a file.

#### 1.1.2.2 Eclipse with WTP

These steps describe how to deploy a custom Java Oracle WebCenter Interaction Development Kit (IDK) project in Eclipse with Web Tools Platform (WTP) installed.

These instructions use Apache Tomcat as an example.

1. Define the server in Eclipse:
    a. Click **File** > **New** > **Other** > **Server** > **Server** and click **Next**.
    b. Select the server type (Apache Tomcat v5.0) and click **Next**.
    c. Select the Apache Tomcat v5.0 installation directory and click **Next**.
    d. Add your custom project to the list of configured projects and click **Finish**.
2. Run and debug the application:
    a. In Project Explorer, right-click your custom project and click **Debug As** > **Debug On Server**.
    b. Select the existing server and click **Finish**.
3. Content services, identity services and custom preference (SCI) pages require additional configuration. You must add the custom class to the appropriate *Impl keys in the web.xml file in the WEB-INF directory. For details, see XXX_missing x-ref to ref_idk_deploymentimplkeys.dita_XXX.
4. When Apache Tomcat starts in a new **Servers** tab, hit http://localhost:8080/<projectname>/servlet/AxisServlet to ensure that Axis has deployed correctly and the web service APIs are correctly configured.

### 1.1.3 Java: Debugging a Custom Oracle WebCenter Interaction Development Kit (IDK) Project

After you create a custom Oracle WebCenter Interaction Development Kit (IDK) project, you must deploy it in your Java application server.

These instructions use Apache Tomcat as an example.

1. Define the server in Eclipse:
    a. Click **File** > **New** > **Other** > **Server** > **Server**and click **Next**.
    b. Select the server type as Apache Tomcat v5.0 and click **Next**.
    c. Select the Apache Tomcat v5.0 installation directory and click **Next**.
    d. Add your project to the list of configured Apache Tomcat projects and click **Finish**.
2. Content services, identity services and SCI pages require additional configuration. You must add the custom class to the appropriate *Impl keys in the web.xml file in the WEB-INF directory. For details on Impl keys, see XXX_missing x-ref to ref_idk_deploymentimplkeys.dita_XXX.
3. Run and debug the application:
    a. In Eclipse Project Explorer, right-click your project and click **Debug As** > **Debug On Server**.
    b. Select the existing server and click **Finish**.
4. When Apache Tomcat starts in a new **Servers** tab, hit `http://localhost:8080/<project name>/servlet/AxisServlet` to

ensure that Axis has deployed correctly and the web service APIs are correctly configured.

### 1.1.4 .NET: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Visual Studio

These steps describe how to set up a custom .NET Oracle WebCenter Interaction Development Kit (IDK) project in Visual Studio.

> **Note:** These instructions assume you have installed the **.NET** version of the Oracle WebCenter Interaction Development Kit (IDK).

1. Start Visual Studio and click **File** > **New Project** > **C# Projects** > **ASP.NET Web Service**.
2. Type an intuitive name in the **Location** field.
3. Delete Service1.asmx and Web.config.
4. In the new project, click **File** > **Add Existing Item**.
5. Browse to the **\devkit** folder in the IDK installation directory.
6. In the **File Types** mask, click **All Files**.
7. Select all the .asmx files and Web.config. Do not select the \bin directory.
8. Click **Open**. You will be prompted to create a class file for each .asmx file; click **No** for each file.
9. In the Solution Explorer (usually in the upper right), you should see the project you created in step 1. Add the IDK assemblies:
    a. Right-click **References** and click **Add Reference**.
    b. Browse to the **\devkit\bin** folder in the IDK installation directory.
    c. Select the assemblies to add to the bin directory: all the .dll files (Ctrl+A). These are the assemblies that resolve the references in the *.asmx files.
        - If you are using the standard (un-signed) version of the IDK, select all the .dll files (Ctrl+A).
        - If you are using the signed dll version of the IDK, select only Plumtree.openlog-framework_signed.dll. (You must deploy the other assemblies in the GAC as described in step f below.)
    d. Click **Open** > **OK**.
    e. In the Solution Explorer References, confirm that you now see idk, openfoundation, etc.
    f. If you are using the signed dll version of the IDK, deploy the following assemblies in the GAC:
        - Plumtree.EDK_signed.dll
        - OpenFoundation_signed.dll
        - Plumtree.openkernel_signed.dll
        - Plumtree.openlog-framework_signed.dll
        - Plumtree.pmb_signed.dll

  - Plumtree.RAT_signed.dll

10. Click **File** > **Add New Item** to create new classes and complete your project.

### 1.1.5 .NET: Deploying a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in IIS

These steps describe how to deploy a custom .NET Oracle WebCenter Interaction Development Kit (IDK) project in IIS.

These instructions assume you have set up Visual Studio for IDK development as described in the previous section, Section 1.1.4, ".NET: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Visual Studio".

1. Compile the class that implements the Oracle WebCenter Interaction Development Kit (IDK) interface(s).
2. Content services, identity services and SCI pages require additional configuration. You must add the class and the assembly that contains it to the appropriate *Assembly and *Impl keys in the web.config file in your project. For details, see XXX_missing x-ref to ref_idk_deploymentimplkeys.dita_XXX.
3. If you do not already have a virtual directory in IIS for your services, add one using the steps below:
   a. Navigate to **Internet Services Manager (Internet Information Services)** in the Control Panel under Administrative Tools.
   b. Select **Default Web Site.**
   c. Click **Action** > **New** > **Virtual Directory** and type the name of your Visual Studio location.
   d. Click **Next** twice. Type the path to the home directory for the IDK: <installdir>\idk\6.0\devkit\dotnet.
   e. Check both the **Read** and **Scripts only** checkboxes if they are cleared (they should be checked by default). Click **Next** then click **Finish**.
4. Copy the compiled class files to the \bin folder in the <installdir>\idk\<version>\devkit\dotnet directory.

## 1.2 Oracle WebCenter Interaction Logging Utilities

Oracle WebCenter Interaction Logging Utilities are a collection of debugging and logging solutions available for use in Oracle WebCenter Ensemble.

Oracle WebCenter Interaction Logging Utilities allow for a wide variety of logging solutions. The Oracle WebCenter Interaction Development Kit (IDK) provides a remote API that allows you to send logging messages from remote web applications.

This chapter contains the following sections:

- Section 1.2.1, "Configuring Oracle WebCenter Interaction Development Kit (IDK) Logging": Oracle WebCenter Interaction Development Kit (IDK) logging is not enabled by default. You can enable logging options programmatically or using the web.xml or Web.config file distributed with the IDK.
- Section 1.2.2, "Using the Oracle WebCenter Interaction Development Kit (IDK) Logging API": The Oracle WebCenter Interaction Development Kit (IDK) logging API allows you to send log messages from remote services and applications to a

variety of logging receivers. This section explains how to use the logging API from Java and .NET applications and from the command line.

## 1.2.1 Configuring Oracle WebCenter Interaction Development Kit (IDK) Logging

To enable and configure Oracle WebCenter Interaction Development Kit (IDK) logging, first determine how the IDK is deployed.

Oracle WebCenter Interaction Development Kit (IDK) logging is disabled by default. If logging is enabled, it is sent only to the local machine by default, requiring direct access to the machine to view the logs. These default settings were chosen to secure potentially sensitive information present in log messages.

- If the Oracle WebCenter Interaction Development Kit (IDK) is deployed as a **Web application** to support Integration Service implementations, edit the distributed Web application configuration file (web.xml or Web.config). For details, see Section 1.2.1.1, "Configuring Java Oracle WebCenter Interaction Development Kit (IDK) Logging (web.xml)" or Section 1.2.1.2, "Configuring .NET Oracle WebCenter Interaction Development Kit (IDK) Logging (Web.config)"
- If the Oracle WebCenter Interaction Development Kit (IDK) is deployed as a **library** supporting a Web application (for example, a pagelet), copy and paste the configuration parameters from the IDK's distributed web.xml/Web.config into your Web application configuration file. For details, see Section 1.2.1.1, "Configuring Java Oracle WebCenter Interaction Development Kit (IDK) Logging (web.xml)" or Section 1.2.1.2, "Configuring .NET Oracle WebCenter Interaction Development Kit (IDK) Logging (Web.config)".
- If the Oracle WebCenter Interaction Development Kit (IDK) is deployed as a **stand-alone application** outside a Web application context, such as report-generating or data loading and dumping applications using the PRC, use programmatic configuration to initialize logging parameters. Programmatic logging configuration can be done at startup, or by using a static initialization call on a façade class that the Web application runtime code uses to obtain logging components or logger instances. For details, see Section 1.2.2, "Using the Oracle WebCenter Interaction Development Kit (IDK) Logging API".

To use the Oracle WebCenter Interaction Development Kit (IDK) Logging API, you must configure the logging receiver to read logs from the IDK. To configure the log receiver, you must know the logging application name. The Oracle WebCenter Interaction Development Kit (IDK) logging application name is configured in the Web application configuration file or set via the `initialize()` method in the Logging API.

> **Note:** Verbose logging cannot be enabled programmatically; you must change a setting in the web.xml or Web.config file.

### 1.2.1.1 Configuring Java Oracle WebCenter Interaction Development Kit (IDK) Logging (web.xml)

For web services using the Java Oracle WebCenter Interaction Development Kit (IDK), the web.xml file is the standard way to configure log instrumentation.

The example below shows the logging settings only. The bulk of the web.xml file has been omitted; environment keys are inserted at the end according to the DTD.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE web-app
```

```
PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN
'http://java.sun.com/j2ee/dtds/web-app_2.2.dtd'>
<web-app>
...
<env-entry>
    <env-entry-name>ptedk.VerboseLogging</env-entry-name
    <env-entry-value>true</env-entry-value>
    <env-entry-type>java.lang.Boolean</env-entry-type>
</env-entry>
<env-entry>
    <env-entry-name>ptedk.LoggingApplicationName</env-entry-name>
    <env-entry-value>EDK</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
    <env-entry-name>ptedk.LogToNetwork</env-entry-name>
    <env-entry-value>true</env-entry-value>
    <env-entry-type>java.lang.Boolean</env-entry-type>
  </env-entry>
<web-app>
```

#### 1.2.1.2 Configuring .NET Oracle WebCenter Interaction Development Kit (IDK) Logging (Web.config)

If you are running the .NET Oracle WebCenter Interaction Development Kit (IDK) as a web application that hosts Web services (that do not use the logging API), the Web.config file is the best way to configure log instrumentation.

The Oracle WebCenter Interaction Development Kit (IDK) Web.config follows the normal precedence rules of IIS Web.config: within a web application, machine.config is read first for configuration values, then overlaid with Web.config from each parent directory within the web application subtree down to the directory containing the running code. The example below shows the logging settings only. All .NET Web applications have Web.config files. If the configuration file does not have an <appSettings> section, it can be added along with the key-value pairs to configure logging.

```
<configuration>
  <appSettings>
      <add key="ptedk.LoggingApplicationName" value="Bulk-Document-Loader" />
      <add key="ptedk.LogToNetwork" value="true" />
      <add key="ptedk.VerboseLogging" value="true" />
   </appSettings>
   <system.web>
  ...
   </system.web>
</configuration>
```

For stand-alone .NET applications outside a web application context, use programmatic configuration. For details, see Section 1.2.2.2, "Using Oracle WebCenter Interaction Development Kit (IDK) Logging in .NET".

#### 1.2.1.3 Oracle WebCenter Interaction Development Kit (IDK) Logging Levels

This page summarizes logging levels and their implementation in Oracle WebCenter Interaction Development Kit (IDK) logging.

The Oracle WebCenter Interaction Development Kit (IDK) `ILogger` interface provides access to all eight standard logging levels.

*Table 1–1 Severity-Based Logging Levels*

| Logging Level | Description | IDK Implementation |
|---|---|---|
| **Debug** | The most common and numerous log call, used for detailed call tracing and parameter logging. The message should contain a detailed descriptive message reporting a minor step taken in the code or providing variable values (or both). | Remote call tracing. Function parameters. ToString() of pagelet settings or service request. |
| **Info** | Used for normal but significant events. Reports a common operation that is of possible interest, for example, serving a new user request or making a remote procedure call. | New pagelet or service request. PRC session initialization (login). The IDK logging service sends an Info message to the "EDK main" logging component when it is initialized. |
| **Warn** | Used for minor problems. Indicates a possible problem which the person responsible for the application should be aware of. | Expected (application) exceptions. For a pagelet, this includes non-proxied requests and missing settings. The Oracle WebCenter Interaction Development Kit (IDK) logging service sends a Warn message to the 'EDK main' logging component when it is initialized if verbose logging is enabled, since the network or application administrator should be aware of possible security implications of sending remote call parameters to a cleartext logging channel. |
| **Error** | Used for major problems affecting application function. Indicates a definite problem that should to be corrected. The message should state and explain the problem and suggest how to fix it. | Unexpected platform exceptions. For a pagelet, this includes errors parsing CSP headers. |
| **Fatal** | Used for problems so severe that the application cannot continue to function. The message should state the problem, explain why it is a problem, and suggest how to fix it. Examples include inability to obtain necessary system or network resources. | A Fatal message is logged when an instance of the class configured for the Web Service object cannot be instantiated. Otherwise reserved for application developer use. |

*Table 1–2 Supplemental Logging Levels*

| Logging Level | Description | IDK Implementation |
|---|---|---|
| **Action** | Used for significant actions (between Info and Warn in severity). Examples include the beginning or ending of a startup routine or the start or completion of a new user request. | Initialize an application component or a new remote session. |
| **Function** | Used to bracket the beginning and ending of a function. Use at the very beginning and end of methods to illustrate code paths and provide context for messages occurring between the beginning and ending function messages. | Dispatching and receiving a remote call, and parsing request parameters. |
| **Performance** | Provides a millisecond timestamp (for example, operation X took # milliseconds). Use to measure operations that may be costly in time. Typically a pair of begin and end performance calls will bracket a blocking call to an operation of interest such as a disk read or write, remote call, external process invocation, database query, or large sort operation. | PRC remote calls. Web request lifecycle for services. |

#### 1.2.1.4 Oracle WebCenter Interaction Development Kit (IDK) Logging API Web Application Variables

To enable Oracle WebCenter Interaction Development Kit (IDK) logging, you must enter the application name and change the settings in the web.xml (Java) or Web.config (.NET) file distributed with the Oracle WebCenter Interaction Development Kit (IDK). The table below lists the applicable variables.

*Table 1–3 Oracle WebCenter Interaction Development Kit (IDK) Logging API Web Application Variables*

| Setting | Default Value | Description |
|---|---|---|
| **ptedk.LoggingApplicationName** | **""** (No logging occurs if the application name is not set.) | OpenLog and Logging Spy use a text string (OpenLog: 'Application' / PTSpy: 'server') to identify a specific log channel to which log appenders can send messages, and from which log receivers can receive messages. To receive messages sent to an OpenLog channel, a listening application must be configured with the same application name used by the log-generating application. To receive log messages from an existing Oracle WebCenter Interaction Development Kit (IDK) deployment in a Web application, set values for the name and logging options according to the example in the web.xml or Web.config file. To receive log messages from a non-Web application that uses the Oracle WebCenter Interaction Development Kit (IDK) (for example, batch or utility processes using remote APIs), set the logging application name programmatically. Use the value in the key ptedk.LoggingApplicationName to set a matching server name in the logging receiver. Note: If the application is already using OpenLog and also using the Oracle WebCenter Interaction Development Kit (IDK), the code must not attempt to initialize OpenLog with a different application name |
| **ptedk.LogToNetwork** | **false** (Logs to local machine only.) | Logging to the network is disabled by default. In this condition, log messages can only be received by OpenLog receiver processes on the local machine, including Logging Spy, the File Logger, or receivers using the OpenLog-Log4J Bridge.Logging can be enabled by setting the value associated with ptedk.LogToNetwork to true in the Web application configuration file. For non-Web applications, you can enable network logging programmatically using the Oracle WebCenter Interaction Development Kit (IDK). |
| **ptedk.VerboseLogging** | **false** (Does not log method parameters or return values unless requested.) | Verbose logging is disabled by default. Basic logging messages are still sent to the log receiver. The pagelet API sends an Info log message with each new pagelet context created (each pagelet request). Any exceptions, errors, or requests for missing settings are logged as Error or Warning as appropriate.If you enable verbose logging, additional messages and details are sent to the log receiver. The pagelet API sends a Warning message informing the log reader that sensitive information may be logged in cleartext. With each pagelet request, the pagelet API sends a Debug message with a toString() of the PortletRequest object, containing request parameters and pagelet settings; and a Debug message with a toString() of the PortletUser object, containing user settings. |

### 1.2.2 Using the Oracle WebCenter Interaction Development Kit (IDK) Logging API

The Oracle WebCenter Interaction Development Kit (IDK) logging API allows you to send log messages from remote services and applications to a variety of logging receivers.

The com.plumtree.remote.logging package provides two interfaces:

- `LogFactory` provides static methods to configure logging, query configuration properties, and obtain `ILogger` instances.
- `ILogger` allows you to test if various log levels are enabled and provides logging methods. To create a logger object, call `LogFactory.getLogger()`.

To use the Oracle WebCenter Interaction Development Kit (IDK) Logging API, you must configure the logging receiver to read logs from the IDK. To configure the log receiver, you must know the logging application name. The Oracle WebCenter Interaction Development Kit (IDK) logging application name is configured in the Web application configuration file or set via the `initialize()` method in the Logging API.

For details on using the logging API, see the following sections:

- Section 1.2.2.1, "Using Oracle WebCenter Interaction Development Kit (IDK) Logging in Java"
- Section 1.2.2.2, "Using Oracle WebCenter Interaction Development Kit (IDK) Logging in .NET"
- Section 1.2.2.3, "Using Oracle WebCenter Interaction Development Kit (IDK) Logging from the Command Line"

### 1.2.2.1 Using Oracle WebCenter Interaction Development Kit (IDK) Logging in Java

This example demonstrates how to enable and use Oracle WebCenter Interaction Development Kit (IDK) logging in a remote Java application.

1. The first step in this example is to enable logging programmatically, by defining the logging application name and setting the log to network option to true. For details on logging options, see Section 1.2.1.1, "Configuring Java Oracle WebCenter Interaction Development Kit (IDK) Logging (web.xml)".

```
import com.plumtree.remote.logging.ILogger;
import com.plumtree.remote.logging.LogFactory;

public class LoggingExample extends Thread
{
   private static final String INSTANCES_COMPONENT_NAME = 'Instances';
   private static final String MAIN_LOOP_COMPONENT_NAME = 'Main Loop';

   // set the application name
   // (legal characters: ASCII alphanumerics plus . - _ and space)
   public static final String LOGGING_APPLICATION_NAME = 'Logging_API_
Example-1';

   // set to true to multicast log messages to local network
   // set to false to send message only listeners on local machine
   public static final boolean LOG_TO_NETWORK = true;

   private ILogger logger;   //instance logging class
   private static ILogger mainLogger;   // main component logging class
```

2. Initialize `LogFactory`. The recommended way to initialize non-web applications is in a static block in the application's main class or a logging utility class. Always check to see if `LogFactory` has already been initialized (for example, as part of an IDK-based web application).

```
if (!LogFactory.isInitialized())
{
   LogFactory.initialize(LOGGING_APPLICATION_NAME, LOG_TO_NETWORK);
}
System.out.print('Set your logging receiver to the \'server\' or \'application
name\' ');
System.out.println(LogFactory.getApplicationName());
System.out.println('The logging component names are \'EDK\', \'' + MAIN_LOOP_
COMPONENT_NAME + '\' and \''
+ INSTANCES_COMPONENT_NAME + '\'.');

mainLogger = LogFactory.getLogger(MAIN_LOOP_COMPONENT_NAME,
LoggingExample.class);
```

This code creates the following messages in Logging Spy. These messages are sent automatically by the Oracle WebCenter Interaction Development Kit (IDK). For the sample code above, the <appname> would be "Logging_API_"

*1 <#> <app name> <date/time> Info EDK main LogFactory Initiating EDK logging on behalf of EDK: LogFactory.*

*2 <#> <app name> <date/time> Info EDK main LogFactory Verbose logging of internal EDK classes is off. It may be enabled by setting ptedk.VerboseLogging='true' .*

3. Create an instance of `ILogger` by calling `LogFactory.getLogger`. In the code below, the `LoggingExample` method sends an Info level log message when an instance is created. The snippet below also uses `ILogger.functionBegin` and `ILogger.functionEnd` to log when a method is entered and exited, `ILogger.action` to log significant events, and `ILogger.performanceBegin` and `ILogger.performanceEnd` to log the time required to execute the methods.

```
public LoggingExample(String instanceName)
{
   setName(instanceName);
   this.logger = LogFactory.getLogger(INSTANCES_COMPONENT_NAME,
LoggingExample.class);
   mainLogger.info('Created new instance named {0}', instanceName);
}
public static void main(String[] args)
{
   final String methodName = 'main';
   mainLogger.functionBegin(methodName);

   // get a timestamp to measure performance of this function
   long performanceStartTicks = mainLogger.performanceBegin();

   mainLogger.action('Creating and starting instances');

   LoggingExample bill = new LoggingExample('Bill');
   bill.start();
   LoggingExample larry = new LoggingExample('Larry');
   larry.start();

   mainLogger.action('Done creating instances');

   // send log message with time since performanceBegin
   mainLogger.performanceEnd(methodName, performanceStartTicks);

   mainLogger.functionEnd(methodName);
}
```

This code creates the following messages in Logging Spy.

*3 <#> <app name> <date/time> Function Main Loop main LoggingExample Entering Function main*

*4 <#> <app name> <date/time> Action Main Loop main LoggingExample Creating and starting instances*

*5 <#> <app name> <date/time> Info Main Loop main LoggingExample Created new instance named Bill*

*6 <#> <app name> <date/time> Info Main Loop main LoggingExample Created new instance named Larry*

*7 <#> <app name> <date/time> Action Main Loop main LoggingExample Done creating instances*

*8 <#> <app name> <date/time> Performance Main Loop main LoggingExample main took 0 ms.*

*9 <#> <app name> <date/time> Function Main Loop main LoggingExample Leaving Function mainInfo*

**4.** The code below demonstrates available logging levels and provides an example of how to use token substitution in formatting strings to construct messages. The thread runs through a small test of logging messages and transfers work to the next by calling `yield()`. Note: Wrap any complex message construction in a conditional block to avoid doing work if there are no listeners at that log level.

```
public void run()
{
   String levelDescriptionFormat = '{0} level messages are {1} by default in 
the log receiver.';
   logger.debug(levelDescriptionFormat, 'Debug', 'off');
   logger.info(levelDescriptionFormat, 'Info',  'off');
   logger.warn(levelDescriptionFormat, 'Warn',  'on');
   logger.error(levelDescriptionFormat, 'Error', 'on');
   logger.fatal(levelDescriptionFormat, 'Fatal', 'on');

   yield();

   // Exceptions may also be caught and logged, and may use token substitution
   try
   {
      throw new InterruptedException(getName() + ' was interrupted.');
   }
   catch (Exception eCaught)
   {
      logger.warn(eCaught, 'Caught an exception from {0}. ', 
eCaught.getClass().getPackage().getName());
   }
}
```

This code creates the following messages in Logging Spy:

*10 <#> <app name> <date/time> Function Instances Larry LoggingExample Entering Function run*

*11 <#> <app name> <date/time> Action Instances Bill LoggingExample Action log messages are on by default in the log receiver.*

*12 <#> <app name> <date/time> Debug Instances Bill LoggingExample Debug level messages are off by default in the log receiver.*

*13 <#> <app name> <date/time> Info Instances Bill LoggingExample Info level messages are off by default in the log receiver.*

*14 <#> <app name> <date/time> Warning Instances Bill LoggingExample Warn level messages are on by default in the log receiver.*

*15 <#> <app name> <date/time> Error Instances Bill LoggingExample Error level messages are on by default in the log receiver.*

*16 <#> <app name> <date/time> Fatal Instances Bill LoggingExample Fatal level messages are on by default in the log receiver.*

*17 <#> <app name> <date/time> Action Instances Larry LoggingExample Action log messages are on by default in the log receiver.*

*18 <#> <app name> <date/time> Debug Instances Larry LoggingExample Debug level messages are off by default in the log receiver.*

*19 <#> <app name> <date/time> Info Instances Larry LoggingExample Info level messages are off by default in the log receiver.*

*20 <#> <app name> <date/time> Warning Instances Larry LoggingExample Warn level messages are on by default in the log receiver.*

*21 <#> <app name> <date/time> Error Instances Larry LoggingExample Error level messages are on by default in the log receiver.*

*22 <#> <app name> <date/time> Fatal Instances Larry LoggingExample Fatal level messages are on by default in the log receiver.*

*23 <#> <app name> <date/time> Warning Instances Bill LoggingExample Caught an exception from - java.lang. java.lang.InterruptedException: Bill was interrupted. - java.lang.InterruptedException: Bill was interrupted. at - com.plumtree.remote.logging.example.LoggingExample.run(LoggingExample.java:110)*

*24 <#> <app name> <date/time> Warning Instances Larry LoggingExample Caught an exception from - java.lang. java.lang.InterruptedException: Larry was interrupted. - java.lang.InterruptedException: Larry was interrupted. at - com.plumtree.remote.logging.example.LoggingExample.run(LoggingExample.java:110)*

#### 1.2.2.2 Using Oracle WebCenter Interaction Development Kit (IDK) Logging in .NET

This example demonstrates how to use Oracle WebCenter Interaction Development Kit (IDK) logging in a remote .NET application.

1. The first step in this example is to enable logging programmatically, by defining the logging application name and setting the log to network option to true. For details on logging options, see Section 1.2.1.2, "Configuring .NET Oracle WebCenter Interaction Development Kit (IDK) Logging (Web.config)".

```
using System;
using System.Threading;
using Plumtree.Remote.Logging;

public class LoggingCommandLineExample
{
   private static readonly String INSTANCES_COMPONENT_NAME = 'Instances';
   private static readonly String MAIN_LOOP_COMPONENT_NAME = 'Main Loop';

   // set the application name
   // (legal characters: ASCII alphanumerics plus . - _ and space)
   public static readonly String LOGGING_APPLICATION_NAME = 'Logging_API_
Example-1';

   // set to true to multicast log messages to local network
   // set to false to send message only listeners on local machine
   public static readonly bool LOG_TO_NETWORK = true;

   private ILogger logger;   //instance logging class
   private static ILogger mainLogger;   // main component logging class
   // thread for each instance of LoggingCommandLineExample
   private Thread _thread;
```

2. Initialize `LogFactory`. The recommended way to initialize non-web applications is in a static block in the application's main class or a logging utility class. Always check to see if `LogFactory` has already been initialized (for example, as part of an IDK-based web application).

```
if (!LogFactory.isInitialized())
{
   LogFactory.Initialize(LOGGING_APPLICATION_NAME, LOG_TO_NETWORK);
}
Console.Out.WriteLine('Set your logging receiver to the \'server\' or
\'application name\' ');
Console.Out.WriteLine(LogFactory.GetApplicationName());
Console.Out.WriteLine('The logging component names are \'EDK\', \'' + MAIN_
LOOP_COMPONENT_NAME + '\' and \'' +
INSTANCES_COMPONENT_NAME + '\'.');

mainLogger = LogFactory.GetLogger(MAIN_LOOP_COMPONENT_NAME,
typeof(LoggingCommandLineExample));
```

This code creates the following messages in Logging Spy. These messages are sent automatically by the Oracle WebCenter Interaction Development Kit (IDK). For the sample code above, the <app name> entry would be "Logging_API_"

*1 <#> <app name> <date/time> Info EDK main LogFactory Initiating EDK logging on behalf of EDK: LogFactory.*

*2 <#> <app name> <date/time> Info EDK main LogFactory Verbose logging of internal EDK classes is off. It may be enabled by setting ptedk.VerboseLogging='true' .*

3. Create an instance of `ILogger` by calling `LogFactory.getLogger`. In the code below, the LoggingExample method sends an Info level log message when an instance is created. The snippet below also uses `ILogger.functionBegin` and `ILogger.functionEnd` to log when a method is entered and exited, `ILogger.action` to log significant events, and `ILogger.performanceBegin` and `ILogger.performanceEnd` to log the time required to execute the methods.

```
public LoggingCommandLineExample(String instanceName)
{
   _thread = new Thread(new ThreadStart(Run));
   _thread.Name = instanceName;
   this.logger = LogFactory.GetLogger(INSTANCES_COMPONENT_NAME,
typeof(LoggingCommandLineExample));
   mainLogger.Info('Created new instance named {0}', instanceName);
}
[STAThread]
public static void main(String[] args)
{
   String methodName = 'main';
   mainLogger.FunctionBegin(methodName);

   // get a timestamp to measure performance of this function
   long performanceStartTicks = mainLogger.PerformanceBegin();

   mainLogger.Action('Creating and starting instances');

   LoggingExample bill = new LoggingExample('Bill');
   bill.Thread.Start();
   LoggingExample larry = new LoggingExample('Larry');
   larry.Thread.Start();

   mainLogger.Action('Done creating instances');

   // send log message with time since performanceBegin
   mainLogger.PerformanceEnd(methodName, performanceStartTicks);

   mainLogger.FunctionEnd(methodName);
}
```

This code creates the following messages in Logging Spy.

*3 <#> <app name> <date/time> Function Main Loop main LoggingExample Entering Function main*

*4 <#> <app name> <date/time> Action Main Loop main LoggingExample Creating and starting instances*

*5 <#> <app name> <date/time> Info Main Loop main LoggingExample Created new instance named Bill*

*6 <#> <app name> <date/time> Info Main Loop main LoggingExample Created new instance named Larry*

*7 <#> <app name> <date/time> Action Main Loop main LoggingExample Done creating instances*

*8 <#> <app name> <date/time> Performance Main Loop main LoggingExample main took 0 ms.*

*9 <#> <app name> <date/time> Function Main Loop main LoggingExample Leaving Function mainInfo*

4. The code below demonstrates available logging levels and provides an example of how to use token substitution in formatting strings to construct messages. The thread runs through a small test of logging messages and interleaves the messages using `Thread.Sleep`.

> **Note:** Wrap any complex message construction in a conditional block to avoid doing work if there are no listeners at that log level.

```
public void Run()
{
   String methodName = 'run';

   // send log message that function is starting
   logger.FunctionBegin(methodName);

   // get a timestamp to measure performance of this function
   long performanceStartTicks = mainLogger.PerformanceBegin();
   Thread.Sleep(1);    // interleaves work to the other thread

   String levelDescriptionFormat = '{0} level messages are {1} by default in
the log receiver.';
   logger.Debug(levelDescriptionFormat, 'Debug', 'off');
   logger.Info(levelDescriptionFormat, 'Info',  'off');
   logger.Warn(levelDescriptionFormat, 'Warn',  'on');
   logger.Error(levelDescriptionFormat, 'Error', 'on');
   logger.Fatal(levelDescriptionFormat, 'Fatal', 'on');

   Thread.Sleep(1);    // interleaves work to the other thread
   // Exceptions may also be caught and logged, and may use token substitution
   try
   {
      throw new ThreadInterruptedException(_thread.Name + ' was interrupted.');
   }
   catch (Exception eCaught)
   {
      logger.Warn(eCaught, 'Caught an exception from {0}. ',
eCaught.GetType().Name);
   }

   Thread.Sleep(1);    // interleaves work to the other thread

   // send log message with time since performanceBegin
   mainLogger.PerformanceEnd(methodName, performanceStartTicks);

   // send log message that function is ending
   logger.FunctionEnd(methodName);
}
public Thread Thread
{
   get
   {
      return _thread;
   }
}
```

This code creates the following messages in Logging Spy:

*10 <#> <app name> <date/time> Function Instances Larry LoggingExample Entering Function run*

*11 <#> <app name> <date/time> Action Instances Bill LoggingExample Action log messages are on by default in the log receiver.*

*12 <#> <app name> <date/time> Debug Instances Bill LoggingExample Debug level messages are off by default in the log receiver.*

*13 <#> <app name> <date/time> Info Instances Bill LoggingExample Info level messages are off by default in the log receiver.*

*14 <#> <app name> <date/time> Warning Instances Bill LoggingExample Warn level messages are on by default in the log receiver.*

*15 <#> <app name> <date/time> Error Instances Bill LoggingExample Error level messages are on by default in the log receiver.*

*16 <#> <app name> <date/time> Fatal Instances Bill LoggingExample Fatal level messages are on by default in the log receiver.*

*17 <#> <app name> <date/time> Action Instances Larry LoggingExample Action log messages are on by default in the log receiver.*

*18 <#> <app name> <date/time> Debug Instances Larry LoggingExample Debug level messages are off by default in the log receiver.*

*19 <#> <app name> <date/time> Info Instances Larry LoggingExample Info level messages are off by default in the log receiver.*

*20 <#> <app name> <date/time> Warning Instances Larry LoggingExample Warn level messages are on by default in the log receiver.*

```
21 <#> <app name> <date/time> Error Instances Larry LoggingExample Error level messages
are on by default in the log receiver.

22 <#> <app name> <date/time> Fatal Instances Larry LoggingExample Fatal level messages
are on by default in the log receiver.

23 <#> <app name> <date/time> Warning Instances Bill LoggingExample Caught an
exception from - java.lang. java.lang.InterruptedException: Bill was interrupted. -
java.lang.InterruptedException: Bill was interrupted. at -
com.plumtree.remote.logging.example.LoggingExample.run(LoggingExample.java:110)

24 <#> <app name> <date/time> Warning Instances Larry LoggingExample Caught an
exception from - java.lang. java.lang.InterruptedException: Larry was interrupted. -
java.lang.InterruptedException: Larry was interrupted. at -
com.plumtree.remote.logging.example.LoggingExample.run(LoggingExample.java:110)
```

### 1.2.2.3 Using Oracle WebCenter Interaction Development Kit (IDK) Logging from the Command Line

These instructions explain how to run the Oracle WebCenter Interaction Development Kit (IDK) Logging API example code (Java or .NET) from the command line.

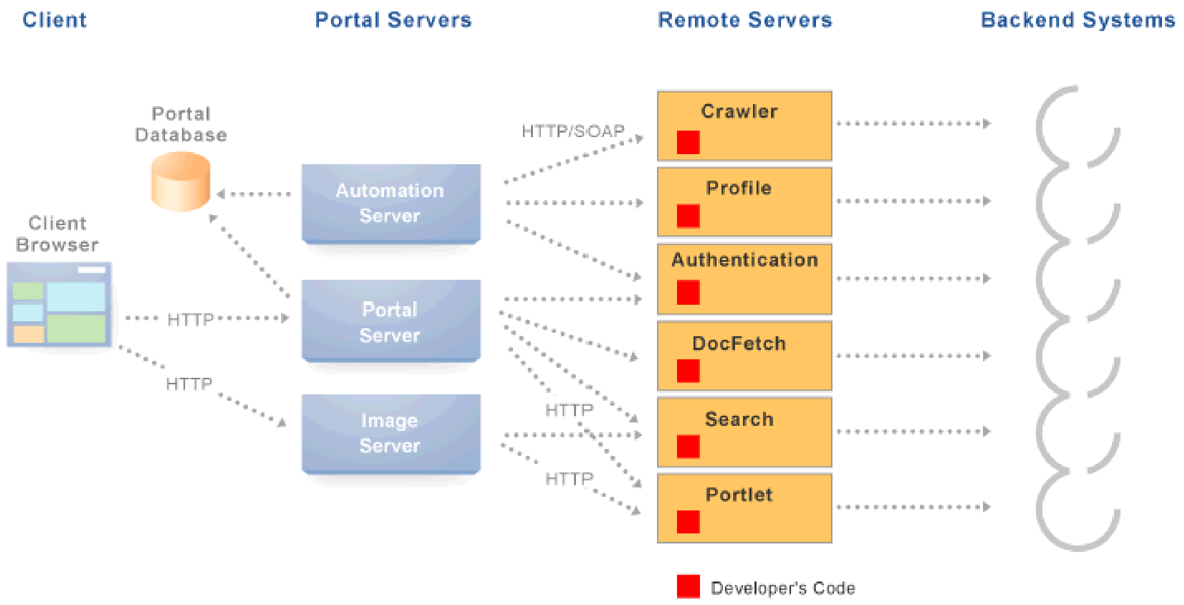
1. Scan the sample code and note the LOGGING_APPLICATION_NAME parameter declared near the top of the class. Change this value if you wish, and record it.
2. Java: Compile with all the idk jar files in the classpath. Make sure servlet.jar and all idk jar files are in the classpath. .NET: Compile the source with reference to idk.dll and its supporting DLLs.
3. Launch Logging Spy. Go to the **Filters** dialog box and add a new server (right-click and select **Add Server** ). Enter the value set for LOGGING_APPLICATION_NAME in the **Add Server** dialog box and click **OK** . Wait a few seconds until a new entry appears in the Filter Settings list .
4. Run the example from the command line. Note any messages displayed in Logging Spy. Error and exception logs are included in the logging demonstration.
5. Go back to the **Filters** dialog in Logging Spy. Click the gray selection box beside the 'server' entry to accept logging for all logging levels.
6. Run the example again. Note that the messages displayed now in Logging Spy include examples of all logging levels, including error and exception logs.

## 1.3 About Server Communication and the Proxy

Oracle WebCenter Ensemble acts as a proxy server, brokering transactions between client computers and external resources.

Services on external resources communicate with Oracle WebCenter Ensemble via HTTP and SOAP as shown in the simplified diagram below. For example, when a browser requests a page, Oracle WebCenter Ensemble makes simultaneous requests to each external resource to retrieve the pagelet content for the page. The external resource reads the current user's preferences from the HTTP headers sent by Oracle WebCenter Ensemble and sends back the appropriate HTML. Oracle WebCenter Ensemble inserts the HTML into the table that makes up the page. Any images stored in the Image Service are retrieved and displayed by the browser.

*Figure 1–1 Server Communication (Simplified Diagram)*



HTTP and SOAP are both necessary because each standard fits the specific needs of different tasks. SOAP involves posting and returning XML documents and is appropriate for exchanging highly structured data. SOAP is used in the server-to-server communication required for content services, identity services, and importing documents. HTTP is a much more lightweight protocol, used in Oracle WebCenter Ensemble for UI presentation, basic configuration and click-through, and caching. For an introduction to SOAP, see Section 1.3.2.4, "About SOAP".

CSP is a platform-independent protocol based on the open standard of HTTP 1.1. The syntax of communication between Oracle WebCenter Ensemble and external resources is defined by CSP. CSP defines custom headers and outlines how services use HTTP to communicate and modify settings. For details on CSP, see Section 1.3.2, "About HTTP and CSP".

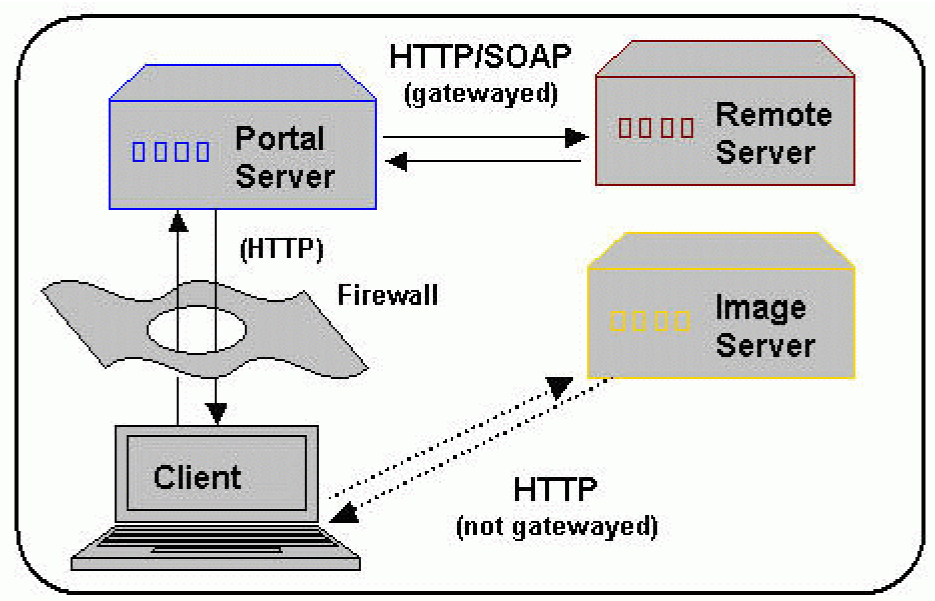
## 1.3.1 The Oracle WebCenter Ensemble Proxy

A proxy server acts as a middleman, brokering transactions between a client computer and another server. This configuration is typically used to serve content to clients that would otherwise be unable to access the external resource, but it can be used to impose additional security restrictions on the client. The proxy hides the external resource; to the end user, the content appears to come directly from the proxy server.

This architecture makes Oracle WebCenter Ensemble the single point of access for content, and allows external resources to reside on a private network or behind a firewall. As long as Oracle WebCenter Ensemble can connect to the external resource, users can view the content, even if they cannot access it directly. To the browser, Oracle WebCenter Ensemble appears to be the source of content on the external resource.

When a user interacts with a service, any request made to a URL in the proxy is automatically rerouted through Oracle WebCenter Ensemble. To the user, the content appears to come from Oracle WebCenter Ensemble; the external resource is an unknown back-end system.

***Figure 1–2 Proxy (Gateway) Architecture***



There are many benefits to this configuration. The most useful to services are:

- **Dynamic functionality and personalization**: Oracle WebCenter Ensemble intercepts requests from pagelets, which allows it to include information stored in the database in HTTP requests and responses. Most of this information is accessible through Oracle WebCenter Interaction Development Kit (IDK) methods. In many situations, an adaptive tag provides the functionality required, including navigation and login elements. Custom tags can be created for additional functionality.
- **Security**: Services can allow users to access content that is not publicly available. Files stored on a secure server can be made available by including specific URLs in the configuration of the proxy. Note: The proxy is a powerful feature, and can compromise security if incorrectly configured. Allowing direct access to a external resource that hosts unprotected private content could create a dangerous security hole.
- **Performance**: Oracle WebCenter Ensemble caches proxied content, decreasing response time for end users and improving performance on the external resource. While proxying works efficiently for content like HTML, it is generally not appropriate for binary data like static images. Images do not need to be transformed, and proxying large images can adversely affect performance. This is one reason the Image Service should be used to prevent routing static images through the proxy.

The collection of URLs that should be proxied for a service is configured in the Resource editor. All URLs that use the Internal URL prefix configured for the resource will be proxied unless **Enable URL Rewriting** is deselected.

Keep the following warnings and best practices in mind when implementing services that use the proxy:

- **URL transformation**: Oracle WebCenter Ensemble must transform code so that proxied URLs open correctly. Before Oracle WebCenter Ensemble sends a response, it parses the HTML and looks for any URLs that use the Internal URL prefix configured for the associated producer resource. Oracle WebCenter

Ensemble transforms any URLs that should be proxied before returning the response to the client. Relative URLs are transformed to point to the correct location.

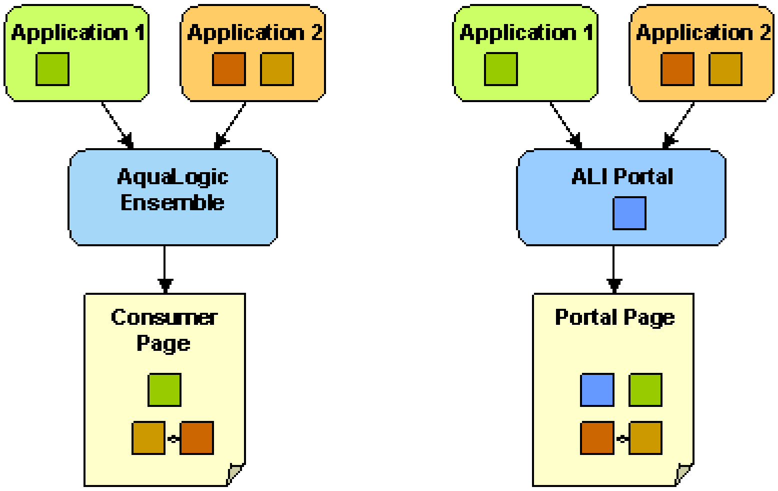
- **Scripting limitations**: JavaScript constructs that dynamically create URLs can cause problems, because they are run after content is already transformed. VBScript is not transformed by the proxy; you can continue to use dynamic scripts and VBScript as long as your code is proxy-aware. To manually mark a URL for transformation, use the pt:url tag. To disable transformation, use pt:transformer with a pt:fixurl attribute of 'off.' For details, see Section 2.2.2, "Adaptive Tags".
- **URL encoding**: It is a best practice to encode all headers that are URLs to prevent unexpected transformation. In JSP, encode all URLs that are written. If the code writes URLs in the body of a page (for example, a link to a preferences page) it should be encoded. The standard Java servlet command response.encodeURL() is the preferred method, but you can also use URLEncoder.encode(url). In the .NET Framework, the HttpUtility.URLEncode class provides the necessary functionality. Note: In .NET, there is no need to encode the redirect URL; this is handled automatically on the back end.

### 1.3.1.1 About Pagelets and the Proxy

All pagelets are designed to be displayed with other pagelets. As explained in the previous section, Oracle WebCenter Ensemble acts as a proxy, processing and combining pagelets from multiple applications to create a single, unified page with a range of functionality.

The code returned by a pagelet is parsed by the proxy server and inserted into the appropriate cell in the HTML table that makes up the mashup page. Pagelets from the same back-end application can interact with each other within the page.

*Figure 1–3 Oracle WebCenter Ensemble as Proxy Server*



In Oracle WebCenter Ensemble, a consumer page defines the layout and includes specific pagelets in the page using adaptive tags. Header navigation can be added using tags.

## 1.3.2 About HTTP and CSP

HTTP is a protocol used mostly for transferring web page content and XML between a server and a client. CSP is a platform-independent protocol based on the open standard of HTTP 1.1 that defines the syntax of communication between Oracle WebCenter Ensemble and external resources.

### 1.3.2.1 HTTP

HTTP communication is made up of Requests and Responses. Requests and Responses are essentially lists of name-value pairs of metadata in headers, along with an optional body. The body is the data that is being transferred (an HTML page or XML file). The metadata in the headers is information about the Request or Response itself (what language the content is in, or how long the browser should cache it). The Request and Response each contain specific information, outlined next. For more detailed information on HTTP, see RFC 2616 (`http://www.faqs.org/rfcs/rfc2616.html`).

The client sends the server an **HTTP Request**, asking for content. The Request body is used only for requests that transfer data to the server, such as POST and PUT.

HTTP Request Format:

```
[METHOD] [REQUEST-URI] HTTP/[VERSION]
[fieldname1]: [field-value1]
[fieldname2]: [field-value2]
[request body, if any]
```

HTTP Request Example:

```
GET /index.html HTTP/1.1
Host: www.plumtree.com
User-Agent: Mozilla/3.0 (compatible; Opera/3.0; Windows 95/NT4)
Accept: */*
Cookie: username=JoeSmith
```

The server sends back an **HTTP Response** that contains page content and important details, such as the content type, when the document was last modified, and the server type. The Response contains an error message if the requested content is not found.

HTTP Response Format:

```
HTTP/[VERSION] [CODE] [TEXT]
[fieldname1]: [field-value1]
[fieldname2]: [field-value2]
[response body, if any (document content here)]
```

HTTP Response Example:

```
HTTP/1.0 200 Found
Last-modified: Thursday, 20-Nov-97 10:44:53
Content-length: 6372
Content-type: text/html
<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 3.2 Final// EN'><HTML>
...followed by document content...
```

Custom HTTP headers can be configured to include specialized information.

> **Note:** Header size limits are controlled by the server that hosts the code. The standard limit for IIS/ASP is 60K. Java Application Servers range from 2K to 10K. These limits are generally configurable; see your server documentation for details.

Services can also access standard HTTP headers, such as the Set-Cookie header or HTTP 1.1 basic authentication header. If you want to investigate HTTP further, you can view all the headers being passed back and forth between your browser and Web server using a tunnel tool. HTTP is used in conjunction with SSL to serve up secure content. Single Sign-On (SSO) also uses HTTP headers for basic authentication.

#### 1.3.2.2 CSP

CSP extends HTTP and defines proprietary headers to pass settings between Oracle WebCenter Ensemble and external resources. CSP outlines how Oracle WebCenter Ensemble services use HTTP to communicate and modify settings. (CSP is also used by Oracle WebCenter Interaction.)

The current version is CSP 1.4, which includes backward compatibility with previous versions. For links to the latest versions of the CSP specification, see Chapter 6, "Additional Development References".

The Oracle WebCenter Interaction Development Kit (IDK) provides simplified, stable interfaces that allow you to write code that communicates using CSP.

#### 1.3.2.3 Oracle WebCenter Ensemble Headers

Oracle WebCenter Ensemble uses a group of custom headers to communicate system and user configuration variables. These headers include information that can be used by services.

All the usefulinformation stored in these headers should be accessed using the OracleWebCenter Interaction Development Kit (IDK). Additional proprietary headers contain the protocol version, proxy type, and aggregationmode. All the key information in these headers is accessible through the `IPortletUser` and `IPortletRequest` interfaces in the IDK.

***Table 1–4 Oracle WebCenter Ensemble Headers***

| Header Name | IDK Method | Description |
|---|---|---|
| User ID | `IPortletUser.GetUserID` | The User ID of the currently logged in user. This value can be used to determine if the session has expired. If UserID=2, the default 'Guest' user is logged in; any other user's session has ended. |
| User Name | `IPortletUser.GetUserName` | The name of the logged in user. The user's name can be used to personalize display or pre-fill form fields. |
| Locale | `IPortletUser.GetUserCharacterSet` | The current user's language and character set. This value is essential when determining the correct content to return in an internationalized implementation. |
| Time Zone | `IPortletRequest.GetTimeZone` | The time zone of the current user in the format used by Oracle WebCenter Ensemble. This value can be used to synchronize external resource time with Oracle WebCenter Ensemble. |
| Image Service URL | `IPortletRequest.GetImageServerURI` | The URL to the root virtual directory of the Image Service in the user's implementation of Oracle WebCenter Ensemble. This location should be used for all static images used in services. |

*Table 1–4 (Cont.) Oracle WebCenter Ensemble Headers*

| Header Name | IDK Method | Description |
|---|---|---|
| Stylesheet URL | `IPortletRequest.GetStylesheetURI` | The URL to the current user's style sheet. In each implementation of Oracle WebCenter Ensemble, the UI is customized. In some portals, users can choose between a selection of stylesheets. Using these styles ensures that pagelets appear in the style of the current user's implementation of Oracle WebCenter Ensemble. |
| Page ID | `IPortletRequest.GetPageID` | The Page ID for the current portal page. This value allows a single pagelet to display different content on different pages. |
| Portlet ID | `IPortletRequest.GetPortletID` | The ID for the current pagelet. This value is useful for appending to the names of HTML forms and client-side JavaScript functions to ensure unique form and function names on the page to avoid name conflicts. |
| Return URL | `IPortletRequest.GetReturnURI` | The URL to the page that the pagelet should return to when finished, usually the page that hosts the pagelet. Preference pages need this URL to return the user to the correct page after settings are configured. |
| Content Mode | `IPortletRequest.GetPortletMode` | The current content mode. This value is used to display pagelet content in the appropriate manner. |
| Browser Type | `IPortletRequest.GetUserInterface` | The type of device being used to access Oracle WebCenter Ensemble. Oracle WebCenter Ensemble can support wireless handheld devices that communicate with HDML, WML, or HTML. |

#### 1.3.2.4 About SOAP

SOAP is a text-based protocol to wrap XML data for any type of transport, providing an efficient way to communicate structured data.

The SOAP 1.1 specification describes SOAP as follows: "SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses."

SOAP is based on web standards. Like HTML, SOAP uses tags to indicate the role of each piece of information. In most implementations, SOAP uses HTTP for its transport protocol. A SOAP request is an XML document that describes a method to invoke on a remote machine and any parameters to be used. A program sends a SOAP request to a SOAP server. The SOAP server tries to execute the method with the parameters it was passed, and it sends back a SOAP response (the result or an error message). A SOAP endpoint is an HTTP-based URL identifying a target for method invocation.

A common analogy illustrates this concept well. If your XML code was a letter, SOAP would be the envelope; like an envelope, SOAP protects content from unauthorized access and provides information about the sender and the addressee. All the elements of the SOAP envelope are defined by a schema. The schema URI is also the identifier for the SOAP envelope namespace: `http://schema.xmlsoap.org/soap/envelope`.

As in standard XML, SOAP uses namespaces to segregate content. The formal designation of a namespace is a URI, usually a URL. Namespaces ensure unique element references, and they allow a processor to pick out which instructions it should obey and treat instructions for other processors as simple data. Processors are set up to handle elements from a particular namespace. Elements that have no namespace are treated as data.

SOAP Message in HTTP Request:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset='utf-8'
Content-Length: nnnn
SOAPAction: 'Some-URI'

<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m='Some-URI'>
<symbol>DIS</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message in HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset='utf-8'
Content-Length: nnnn

<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m='Some-URI'>
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Download the complete SOAP 1.1 specification from the World Wide Web Consortium at `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`.

The Oracle WebCenter Ensemble SOAP API exposes commonly used elements of the traditional Oracle WebCenter Ensemble API, focused on the functions required to develop applications that access portal users, communities, pagelets, and directory functions. The Oracle WebCenter Interaction Development Kit (IDK) PRC API provides an efficient, object-oriented way to call into Oracle WebCenter Ensemble's SOAP API. For details, see Section 2.1.3, "Using Programmable Remote Client (PRC) Remote APIs".

# 2

# Oracle WebCenter Ensemble Pagelet Development

The following sections provide general information about Oracle WebCenter Ensemble pagelet development and configuration.

- Section 2.1, "Oracle WebCenter Interaction Development Kit (IDK) Proxy API": The bea.alui.proxy package provides access to information about the environment in which the pagelet is displayed and the user currently accessing the pagelet, including session preferences associated with that user. This package also includes methods to implement security and access XML payloads.
    - The IDK includes a collection of Remote APIs that provide access to functionality within Oracle WebCenter Interaction, Oracle WebCenter Collaboration, and the portal Search Service. These APIs are supported by Oracle WebCenter Ensemble, and can be used by any pagelet deployed in an environment with access to these applications. For details, see Section 2.1.3, "Using Programmable Remote Client (PRC) Remote APIs".
- Section 2.2, "Adaptive Pagelets": Adaptive pagelets allow you to create a coordinated page with dynamic, interactive functionality comprised of cross-platform services that talk to multiple back-ends.

  Adaptive pagelet tools include the following:

    - **Adaptive Tags**: Adaptive Tags are used to display contextual data and control Oracle WebCenter Ensemble from remote pagelets. Unlike the Oracle WebCenter Interaction Development Kit (IDK), Adaptive Tags use XML in pagelet content instead of code, which avoids a network round trip. Tags can be included in the markup returned by any proxied page (HTML, JSP or ASP.Net). Using the attributes defined in the tag, the Oracle WebCenter Ensemble proxy transforms the XML and replaces it with standard HTML and/or executes the relevant operations. The Adaptive Tag collection currently includes libraries for use in both Oracle WebCenter Interaction and Oracle WebCenter Ensemble, as well as libraries that are specific to each environment. For details, see Section 2.2.2, "Adaptive Tags"
    - **Oracle WebCenter Interaction Scripting Framework**: The Oracle WebCenter Interaction Scripting Framework is a client-side JavaScript library that provides services to pagelets and proxied pages. For details, see Section 2.2.3, "Oracle WebCenter Interaction Scripting Framework".
- Section 2.3, "Session Preferences": Pagelets can use preferences to communicate with each other, but accessing preferences usually requires a round trip to a database. Session preferences provide a way to store and share settings in the user's session within the client browser.

- Section 2.4, "Pagelet Caching": Caching is the functionality that allows Oracle WebCenter Ensemble to request pagelet content, save the content, and return the saved content to users when appropriate. The importance of caching cannot be overstated.
- Section 2.5, "Pagelet Internationalization": These tips and best practices apply to all pagelets that will be translated into multiple languages.
- Section 2.6, "Pagelet Configuration in Oracle WebCenter Ensemble": To deploy a pagelet in Oracle WebCenter Ensemble, you must configure Resource and Pagelet objects and configure security settings.

## 2.1 Oracle WebCenter Interaction Development Kit (IDK) Proxy API

This section provides an introduction to the Oracle WebCenter Interaction Development Kit (IDK) Proxy API. For more details on objects and methods, see the API documentation. For details on Oracle WebCenter Interaction-specific portlet interfaces, see the *Oracle Fusion Middleware Web Service Developer's Guide for Oracle WebCenter Interaction*.

The bea.alui.proxy package/namespace includes the following interfaces:

- `IProxyContext`
- `IProxyRequest`
- `IProxyResponse`
- `IProxyUser`

In general, these interfaces are called in the following order:

1. A pagelet uses `ProxyContextFactory.getInstance().createProxyContext` to initiate a connection for communicating with Oracle WebCenter Ensemble.
2. The `IProxyContext` object returned allows the pagelet to access information about the request and response, the current user, and the session. The pagelet uses this information as needed, in arbitrary order, to generate a proper response. Using `IProxyContext`, the pagelet can access `IProxyRequest`, `IProxyUser`, `IRemoteSession` and `IProxyResponse`.
3. The pagelet retrieves parameters from the request using `IProxyRequest`.
4. The pagelet retrieves user information and preferences from `IProxyUser`.
5. The pagelet can access functionality in Oracle WebCenter Interaction applications using `IRemoteSession`. For details, see Section 2.2.1, "Adaptive Pagelet Design Patterns".
6. The pagelet constructs a response using `IProxyResponse`. The response includes content to be displayed and any settings to be stored or removed.

For examples of using IProxy interfaces in a pagelet, see Section 2.1.1, "Creating a Custom Pagelet with the Java Oracle WebCenter Interaction Development Kit (IDK) Proxy API" and Section 2.1.2, "Creating a Custom Pagelet with the .NET Oracle WebCenter Interaction Development Kit (IDK) Proxy API".

### 2.1.1 Creating a Custom Pagelet with the Java Oracle WebCenter Interaction Development Kit (IDK) Proxy API

This example creates a simple pagelet that displays information from the proxy, including setting values.

1. Before writing any code, create a new Oracle WebCenter Interaction Development Kit (IDK) project as described in Section 1.1.1, "Java: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Eclipse".
2. In the new project, create a new JSP page for the pagelet (pagelet.jsp).
3. Implement your code. The pagelet code shown below instantiates the Oracle WebCenter Interaction Development Kit (IDK) and uses the `IProxyContext` interface to retrieve `IProxyRequest` and `IProxyUser` objects to access information about the user and the settings associated with the pagelet.

> **Note:** There is no need to include html, head and body tags; the display is handled by the Consumer resource.

```
<%@ page language='java' import='com.bea.alui.proxy.*' %>
<%
String Att1 = 'no setting';
String Att2 = 'no setting';
String sessionVariable = 'no setting';

//get the idk
IProxyContext proxyContext =
ProxyContextFactory.getInstance().createProxyContext(request, response);
IProxyRequest proxyRequest = proxyContext.getProxyRequest()

IProxyUser proxyUser = proxyRequest.getUser();
String userName = proxyUser.getUserName();
int userID = proxyUser.getUserID();

Att1 = proxyRequest.getSetting('Att1')
Att2 = proxyRequest.getSetting('Att2');
sessionVariable = proxyRequest.getSetting('sessionVar');

byte[] payload = proxyRequest.getPayload().getText();
String payloadStr = new String(payload)
%>

<p>User name: <%=userName%><br/>
User ID: <%=userID%><br/>
Attribute 1: <%=Att1%><br/>
Attribute 2: <%=Att2%><br/>
Session variable: <%=sessionVariable%><br/>
Payload: <textarea name=xml cols=80 rows=6> <%=payloadStr%> </textarea>
</p>
```

### 2.1.2 Creating a Custom Pagelet with the .NET Oracle WebCenter Interaction Development Kit (IDK) Proxy API

This example creates a simple pagelet that displays information from the proxy, including setting values. .NET pagelets use a code-behind page (.aspx.cs) to retrieve settings and a Web form (.aspx) to display the pagelet content.

1. Before writing any code, create a new Oracle WebCenter Interaction Development Kit (IDK) project as described in Section 1.1.4, ".NET: Setting Up a Custom Oracle WebCenter Interaction Development Kit (IDK) Project in Visual Studio".
2. In the new project, implement your code. The example below uses a code-behind page and a web form.

The code-behind page (IDKPagelet.aspx.cs) instantiates the Oracle WebCenter Interaction Development Kit (IDK) and uses the `IProxyContext` interface to retrieve `IProxyRequest` and `IProxyUser` objects to access information about the user and the settings associated with the pagelet.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using Plumtree.Remote.Portlet;
using System.Xml;
using System.Text;
using Bea.Alui.Proxy;

namespace IDKProxyWS
{
/// <summary>
/// Hello World Pagelet
/// </summary>
   public class IDKPagelet : System.Web.UI.Page
   {
      public String name;
      public bool isGuest;
      public int userID;
      public String envType;
      public String payload;
      public String Att1,Att2;
      public String SessionVar;
      private void Page_Load(object sender, System.EventArgs e
      {
         // Put user code to initialize the page here
         InitializeCSP();
      }
      private void InitializeCSP()
      {
         IProxyRequest proxyRequest;
         IProxyResponse proxyResponse;
         IProxyUser proxyUser;
         IProxyContext proxyContext;
         ProxyContextFactory factory;
         HttpRequest request = HttpContext.Current.Request;
         HttpResponse response = HttpContext.Current.Response;

         try
         {
            factory = ProxyContextFactory.getInstance();
            proxyContext = factory.CreateProxyContext(request, response);
            proxyRequest = proxyContext.GetProxyRequest();
```

```
                proxyResponse = proxyContext.GetProxyResponse();
                envType = proxyRequest.GetEnvironment().GetType().ToString();
                proxyUser = proxyRequest.GetUser();
                isGuest = proxyUser.IsAnonymous();
                name= proxyUser.GetUserName();
                userID = proxyUser.GetUserID();

                Att1 = (String)proxyRequest.GetSetting('attr1');
                Att2 = (String)proxyRequest.GetSetting('attr2');
                Att2 = (String)proxyRequest.GetSetting('SessionVar');

                byte[] bpayload = proxyRequest.GetPayload().GetText()
                System.Text.ASCIIEncoding enc = new System.Text.ASCIIEncoding()
                payload = enc.GetString(bpayload)
            }
            catch(Bea.Alui.Proxy.NotGatewayedException e)
            {
            }
        }
    }
#region Web Form Designer generated code
...
#endregion
}
```

The Web form that displays the pagelet (IDKPagelet.aspx) displays the information retrieved by the code-behind page above.

```
<%@ Page Language='c#' runat='server' CodeBehind='IDKPagelet.aspx.cs'
AutoEventWireup='false' inherits='IDKProxyWS.IDKPagelet' %>
<%@ import Namespace='System.Collections' %>
<%@ import Namespace='System.Web' %>
<%@ import Namespace='System.Web.UI' %>

<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN' >
<html>
<head>
<title>IDKPagelet</title>
<meta name='GENERATOR' Content='Microsoft Visual Studio .NET 7.1'>
<meta name='CODE_LANGUAGE' Content='C#'>
<meta name='vs_defaultClientScript' content='JavaScript'>
<meta name='vs_targetSchema'
content='http://schemas.microsoft.com/intellisense/ie5'>
</head>

<body MS_POSITIONING='GridLayout'>
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>

Proxy Pagelet <BR>
<%
   Response.Write('IDK Proxy Pagelet<BR>');
   Response.Write('Environment Type ' + envType + '<BR>');
   Response.Write('Guest User? ' + isGuest + '<BR>');
   Response.Write('User Name: ' + name + '<BR>');
   Response.Write('User ID: ' + userID + '<BR>');
   Response.Write('<P>');

   Response.Write('Pagelet Attributes:<BR>');
   Response.Write('Attribute1: ' + Att1 + '<BR>');
   Response.Write('Attribute2: ' + Att2 + '<BR>')
   Response.Write('SessionVar: ' + SessionVar + '<BR>')
   Response.Write('<P>')
   Response.Write('Pagelet XML Payload:<BR>');
   Response.Write('<textarea name=xml cols=80 rows=6>' + payload + '</textarea>');
   Response.Write('<P>');
%>
</span>
</body>
</html>
```

### 2.1.3 Using Programmable Remote Client (PRC) Remote APIs

The plumtree.remote.prc package includes a collection of APIs that provide access to functionality within Oracle WebCenter Interaction, Oracle WebCenter Collaboration, and the portal Search Service. These APIs are supported by Oracle WebCenter Ensemble, and can be used by any pagelet deployed in an environment with access to these applications.

PRC APIs free you from serializing SOAP messages and minimize the amount of data that travels between the portal and other servers, improving performance.

The PRC is included with both the Java and .NET versions of the Oracle WebCenter Interaction Development Kit (IDK). The Java version includes Apache AXIS 1.0; the .NET version uses the platform-native SOAP client stack. Java clients can call .NET portals and vice-versa; the PRC takes care of the communication between AXIS and .NET. Pagelets that use the PRC can be deployed in either Oracle WebCenter Interaction or Oracle WebCenter Ensemble. For details on using the PRC, see the *Oracle Fusion Middleware Web Service Developer's Guide for Oracle WebCenter Interaction*.

## 2.2 Adaptive Pagelets

Adaptive pagelets allow you to create a coordinated page with dynamic, interactive functionality comprised of cross-platform services that talk to multiple back-ends. For detailed examples, see Section 2.2.1, "Adaptive Pagelet Design Patterns".

Adaptive pagelet tools include the following:

- **Adaptive Tags**: Oracle WebCenter Ensemble provides a collection of useful XML tags that can be included in the markup returned by any proxied page, including pagelets. For details, see Section 2.2.2, "Adaptive Tags"
- **Oracle WebCenter Interaction Scripting Framework**: The Oracle WebCenter Interaction Scripting Framework is a client-side JavaScript library that provides services to pagelets and proxied pages. For details, see Section 2.2.3, "Oracle WebCenter Interaction Scripting Framework".

For additional information on adaptive pagelets, see Section 2.2.4, "Adaptive Pagelet Development Tips".

### 2.2.1 Adaptive Pagelet Design Patterns

Adaptive pagelet design patterns provide solutions for broad classes of problems, and provide the base for a range of cross-platform services.

The **Master-Detail** design pattern uses two pagelets; users select an item from a list in one, and the details for that item are retrieved from the external resource and displayed in another. For example, a set of customers could be listed by name in the "master" pagelet. When the user clicks a name in this pagelet, the "detail" pagelet presents details about the item. The detail pagelet for a customer list could list all the important information about that customer, such as name, address, and phone number. This pattern assumes a tight coupling between the two pagelets; both the master pagelet and detail pagelet must be displayed on the same page. For details and sample code, see Section 2.3, "Session Preferences". For a looser coupling, use the Broadcast-Listener pattern.

The **Broadcast-Listener** design pattern is similar to the Master-Detail pattern, but assumes a loose coupling between pagelets. Users can select an item or perform some other action in a "broadcast" pagelet, which causes the content in other related "listener" pagelets to be redrawn. The major difference is that the Broadcast-Listener pattern relies on the Oracle WebCenter Interaction Scripting Framework to raise an event when an action is performed in the "broadcast" pagelet. One or more "listener" pagelets can respond to this event and update their content accordingly. For details and sample code, see Section 2.2.3.2, "Using Oracle WebCenter Interaction Scripting Framework Event Notification".

**In Place Refresh** allows you to refresh the content in a pagelet without refreshing the page. For details and sample code, see Section 2.2.3.3, "Using In-Place Refresh".

The **Structured Response** design pattern handles structured HTTP responses, typically encoded as XML. In many cases it can be expensive and inefficient to send large amounts of HTML back in response to some HTTP request, if only a small part of the user interface needs to be changed. This is especially true with more complex user interfaces. In these cases, the response can be encoded in XML. The client-side response handler can then parse the XML, and update the user interface (or perform some other action) based on that response. Use the Structured Response design pattern to redraw a small area of the user interface after making an HTTP request, or to access a simple HTTP/URI type web service from a pagelet. The example code below (structuredresponse_portlet.html) accesses an RSS feed from a selection of news sites.

```
<!-- jsxml includes -->
<a id="imgServerHref" href="pt://images/plumtree" style="display:none"></a>
<script type="text/javascript"
src="pt://images/plumtree/common/private/js/PTLoader.js"></script>
<script type="text/javascript">
var oImgServer = new Object();
oImgServer.location = document.getElementById('imgServerHref').href;
var imageServerURL = document.getElementById('imgServerHref').href;
var imageServerConnectionURL = oImgServer.location;
new PTLoader(imageServerURL, imageServerConnectionURL).include('jsxml','en');
</script>

<!-- jscontrols includes -->
<link rel="stylesheet" type="text/css"
href="/portal-remote-server/js/jscontrols/styles/css/PTMenu.css"/>
<link rel="stylesheet" type="text/css"
href="/portal-remote-server/js/jscontrols/styles/css/PTRichTextEditor.css"/>
<script type="text/javascript"
src="/portal-remote-server/js/jscontrols/strings/PTControls-en.js"></script>
<script type="text/javascript"
src="/portal-remote-server/js/jscontrols/PTControls.js"></script>

<!-- Inline JS helper functions -->
<!-- NOTE: It is standard practice to use namespace tokens (e.g., <pt:nameSpace
pt:token="$$TOKEN$$" xmlns:pt="http://www.plumtree.com/xmlschemas/ptui/"/>) to
ensure unique global JavaScript function and object names. For simplicity, we do
not do that here.
-->

<script defer type="text/javascript" id="structured-response-portlet-A-script">
// Function that gets the RSS XML feed found at the specified url
```

```
getRSSFeed = function(url)
  {
  // First clear out any existing rows in the table
  channelTable.clearRows();

  // Force the transformer to fix up the url
  var oURL = new Object();
  oURL.location = url;

  // Do the http get
  var get = new PTHTTPGETRequest(oURL.location, handleRSSResponse);
  get.invoke();
  }

// Function that handles the RSS XML response and updates the table based on the
RSS items
handleRSSResponse = function(response)
  {
  // Get the rss xml
  var xml = response.responseText;
  if (!xml || xml.indexOf('<?xml') == -1) { return; }

  // Parse into a dom, and get the channel node
  var xmlDOM = new PTXMLParser(xml);
  var rssNode = xmlDOM.selectSingleNode('rss');
  var channelNode = rssNode.selectSingleNode('channel');

  // Get the channel title and set the status bar text in the table
  var channelTitle = channelNode.selectSingleNode('title').getNodeValue();
  channelTable.statusBarText = '<b>Loaded Channel</b>: ' + channelTitle;

  // Get channel item nodes
  var itemNodes = channelNode.selectNodes('item');

  // Build table rows
  channelTable.rows = new Array();
  for (var i=0; i<itemNodes.length; i++)

    {
    var itemNode = itemNodes[i];

    // Get channel item properties
    var itemTitle = itemNode.selectSingleNode('title').getNodeValue();
    var itemLink = itemNode.selectSingleNode('link').getNodeValue();
    var itemDescription = itemNode.selectSingleNode('description').getNodeValue();
    if (itemNode.selectSingleNode('author'))
      var itemAuthor = itemNode.selectSingleNode('author').getNodeValue();
    if (itemNode.selectSingleNode('category'))
      var itemCategory = itemNode.selectSingleNode('category').getNodeValue();
    if (itemNode.selectSingleNode('pubDate'))
      var itemPubDate = itemNode.selectSingleNode('pubDate').getNodeValue();

    // Create a row and add it to the table
    var row = new PTRow();
    row.parent = channelTable;
    row.id = i;
    row.uid = i;
    row.previewText = itemDescription;
    row.link = itemLink;
    row.columnValues[0] = new PTTextColumnValue(itemTitle);
    row.columnValues[1] = new PTTextColumnValue(itemCategory);
    row.columnValues[2] = new PTTextColumnValue(itemAuthor);
    row.columnValues[3] = new PTTextColumnValue(itemPubDate);
    channelTable.rows[channelTable.rows.length] = row;
    }

  // Redraw the table
  channelTable.draw();
  }
</script>

<b>Select RSS Feed:</b>
<a href="#"
onclick="getRSSFeed('http://www.wired.com/news/feeds/rss2/0,2610,,00.xml'); return
false;">Wired News</a>
<a href="#" onclick="getRSSFeed('http://news.com.com/2547-1_3-0-5.xml'); return
false;">CNET News.com</a>
<a href="#"
onclick="getRSSFeed('http://partners.userland.com/nytRss/nytHomepage.xml'); return
false;">NY Times</a>
<br><br>

<!-- Set up a table control to display channel items -->
<div id="channelTableContainer"></div>
<script defer type="text/javascript">
  var channelTable = new PTTableControl();
  channelTable.locale = 'en_US';
  channelTable.objName = 'channelTable';
  channelTable.container = 'channelTableContainer';
  channelTable.baseURL =
'/imageserver/plumtree/common/private/portal-remote-server/js/jscontrols/1/';
  channelTable.statusBarText = 'No RSS Feed Selected';
  channelTable.rowDetailAction = new
PTJavaScriptAction('window.open(\'${ROW.link}\');');
  channelTable.columns[0] = new PTColumn();
  channelTable.columns[0].name = 'Title';
  channelTable.columns[0].width = '40%';
  channelTable.columns[1] = new PTColumn();
  channelTable.columns[1].name = 'Category';
  channelTable.columns[1].width = '20%';
  channelTable.columns[2] = new PTColumn();
  channelTable.columns[2].name = 'Author';
  channelTable.columns[2].width = '20%';
  channelTable.columns[3] = new PTColumn();
  channelTable.columns[3].name = 'Publication Date';
  channelTable.columns[3].width = '20%';
  channelTable.areColumnsResizable = true;
  channelTable.clientSortEnabled = true;
  channelTable.scrollHeight = 250;

  channelTable.init();
  channelTable.draw();
</script>
</div>
```

### 2.2.2 Adaptive Tags

Oracle WebCenter Ensemble provides a collection of useful XML tags that can be included in the markup returned by any proxied page, including pagelets.

Using the attributes defined in the tag, the proxy transforms the XML and replaces it with standard HTML to be displayed in a browser. For example, when used in a pagelet, the following code is replaced with the date and time in the current user's locale.

```
<pt:standard.currenttime xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
```

The adaptive tag libraries provide access to a wide range of components.

The **core** tag library provides two basic tags to support core tag functionality.

- `pt:core.debugmode` toggles debug mode.
- `pt:core.html` allows you to use HTML tags within JavaScript, and supports attribute replacement.

The tags in the **constants** library provide access to useful URLs, including the stylesheet, Image Service, and the correct return URL for the current user.

***Table 2–1 Tags in the constants Library***

| Tag | Replaced with | Example |
|---|---|---|
| The stylesheet URL in proxied pages and pagelets | `<link type="text/css" href="pt://styles" rel="StyleSheet"></link>` | `pt://styles` |
| The URL to the Image Service | `<img src="pt://images/plumtree/portal/public/img/icon_help.gif">` | `pt://images` |
| A URL that returns users to the page from which they came (the page on which the pagelet that launched the page is hosted) | `<a href="pt://return">Back</a>` | `pt://return` |

The **common** tag library provides access to useful functionality, including URL transformation and namespace definition. This library also allows you to insert error information in the page, and CSS and JavaScript references in the Head element in a proxied HTML page. For details, see Section 2.2.2.5, "Common Adaptive Tag Library (pt:common)".

The tags in the **logic** library handle basic logic, including creating data objects and collections, setting shared variables, and looping over a data collection. For details, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)".

In addition to the tags above, platform-specific tags are available to access additional information and functionality in Oracle WebCenter Ensemble. For details, see Section 2.2.2.4, "Oracle WebCenter Ensemble Adaptive Tag Library (pt:ensemble)".

For important information about using tags, see the following sections:

- Section 2.2.2.1, "Adaptive Tag Development Tips"
- Section 2.2.2.2, "Using Internationalized Strings in Adaptive Tags"
- Section 2.2.2.3, "Using Variables in Adaptive Tags"

For information on how Oracle WebCenter Ensemble processes tags, see Section 2.2.2.7, "About Adaptive Tag Control Flow".

You can also create custom tags; for details, see Section 2.2.2.8, "Creating Custom Adaptive Tags".

For a full list of tags and attributes, see the tagdocs.

#### 2.2.2.1 Adaptive Tag Development Tips

These syntax rules and tips apply to all adaptive tags.

- **All tags are XML compliant.** For example, only strings are allowed; you cannot use a tag within an attribute of another tag (`<legal a=<illegal/>/>`).
- **All adaptive tags belong to the namespace http://www.plumtree.com/xmlschemas/ptui/.** The namespace prefix must be "pt" (xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/). To avoid including the namespace in every tag, enclose all tags in a span that defines the namespace.
- **All adaptive tag attributes require the "pt:" prefix.** If you do not include the pt prefix, Oracle WebCenter Ensemble will not return an error, but will replace the attribute with the default value when the tag is processed.
- **The adaptive tag framework displays tag errors as HTML comments.** If you suspect that a tag error has occurred, simply view the HTML source for the page. If there was a problem, there should be an HTML comment where the adaptive tag would have been.
- **Adaptive tags adhere to XHTML specifications.** These specifications are not handled correctly by all HTML editors and IDEs. Some editors do not display tags correctly because of the required "pt:" prefix before tags.
- **Use tag debug mode for additional insight into tag errors.** Turning on Debug Mode causes the adaptive tag framework to output HTML comments declaring the start and end of each tag. This can be useful for determining whether a tag ran and did not output the expected result, or did not run at all, for example. Note: Standard HTML tags are not wrapped in HTML comments.

#### 2.2.2.2 Using Internationalized Strings in Adaptive Tags

Adaptive tag attribute value replacement allows you to display localized content based on the current user's locale.

Oracle WebCenter Ensemble stores internationalized strings in localized string files with different files for each supported language. Oracle WebCenter Ensemble knows the locale of the current user and retrieves strings from the correct language folder automatically. To internationalize a pagelet, move all strings into custom string files and translate them.

To display content in the pagelet, reference the strings using the value tag from the Logic tag library. As noted above, Oracle WebCenter Ensemble retrieves the string from the correct language folder automatically. For example, the HTML below retrieves the first string from a XML language file called my_message_file.xml.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
  <pt:logic.value pt:value="$#1.my_message_file"/>
</span>
```

For details on tags in the Logic tag library, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)".

#### 2.2.2.3 Using Variables in Adaptive Tags

Adaptive tag attribute value replacement allows you to access data stored in memory.

The following simple example uses the variable and value tags from the Logic tag library to store a value in memory and then display it in HTML.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
  <pt:logic.variable pt:key="test" pt:value="example text"/>
  <pt:logic.value pt:value="$test"/>
</span>
```

Attribute value replacement can also be used to display more complicated memory structures. Data objects can contain multiple name value pairs. The following example creates a data object with the name and URL of a link, and then displays the name.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
  <pt:logic.data pt:key="testdata" url="http://www.myco.com" name="My company"/>
  <pt:logic.value pt:value="$testdata.name"/>
</span>
```

Attribute value replacement cannot be used with tags outside the adaptive tag libraries. However, the pt.core.html tag supports attribute replacement within a tag and allows you to generate any HTML tag. Use the pt:tag attribute to specify the HTML tag and list the necessary HTML attributes as XML attributes. All non-adaptive tag attributes (attributes not prefixed with "pt:") are included automatically in the outputted HTML tag. For example, the following code creates an HTML anchor tag using an in-memory value for the "href" attribute.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:core.html pt:tag="a" href="$myurl" title="My title">My link</pt:core.html>
</span>
```

This code would be transformed to the following HTML: `<a href="[data stored in the $myurl attribute]" title="My title">My link</a>`.

The example below combines several different techniques and tags to show how to loop over a data collection and output HTML. This code outputs several HTML links with lines in between them.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
  <pt:logic.collection pt:key="testcollection">
    <pt:logic.data url="http://www.myco.com" name="My company"/>
    <pt:logic.data url="http://www.otherco.com" name="Other company"/>
  </pt:logic.collection>
  <pt:logic.foreach pt:data="testcollection" pt:var="link">
    <pt:core.html pt:tag="a" href="$link.url">
    <pt:logic.value pt:value="$link.name"/>
    </pt:core.html>
    <pt:logic.separator><br><br></pt:logic.separator>
  </pt:logic.foreach>
</span>
```

For details on Logic tags, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)". For details on using localized strings in tags, see Section 2.2.2.2, "Using Internationalized Strings in Adaptive Tags".

### 2.2.2.4 Oracle WebCenter Ensemble Adaptive Tag Library (pt:ensemble)

The Oracle WebCenter Ensemble tag library (pt:ensemble) provides tags to insert pagelets in Oracle WebCenter Ensemble consumer pages and access authentication and role information for Oracle WebCenter Ensemble resources.

*Table 2–2 Tags in the Oracle WebCenter Ensemble Adaptive Tag Library*

| Tag | Function | More Information |
|---|---|---|
| `pt:ensemble.inject` | Injects the output of the specified pagelet into the page. | Section 2.2.2.4.1, "Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.resourcedata` | Stores data for a specific Oracle WebCenter Ensemble resource, if available, in memory as a data object containing information about the resource. | Section 2.2.2.4.4, "Accessing Resource Data Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.authsourcedata` | Stores a list of available authentication sources for the currently requested resource in memory. | Section 2.2.2.4.2, "Accessing Authentication Data Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.loginlink` | Stores the URL prefix for the login page in memory using the given key and scope. | Section 2.2.2.4.3, "Accessing the Login URL Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.roleexpr` | Evaluates a role expression and stores the result as a boolean in memory. Designed to work with the `pt:logic.if` tag. | Section 2.2.2.4.5, "Accessing User Roles Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.rolelist` | Stores a list of the roles for the current user in memory. | Section 2.2.2.4.5, "Accessing User Roles Using Oracle WebCenter Ensemble Adaptive Tags" |
| `pt:ensemble.ssologout` | Notifies Oracle WebCenter Ensemble that the current user should be logged out of all resources. Used as singleton only (does not display the contents of the tag). | |

#### 2.2.2.4.1 Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags

The `pt:ensemble.inject` tag injects the output of the specified pagelet into the page.

The pagelet is referenced by the fully qualified name of the pagelet as defined in Oracle WebCenter Ensemble, in the form `libraryname:pageletname`. Any non-Oracle WebCenter Ensemble attributes (not prefixed with "pt:") will be passed on to the pagelet. Any HTML content inside the pagelet tag will be passed to the pagelet as an XML payload.

```
<pt:ensemble.inject pt:name="mylibrary:mypagelet" pagelet-attribute="A pagelet
attribute">
<?xml version="1.0" encoding="utf-8"?>
<doc>This is an XML payload.</doc>
</pt:ensemble.inject>
```

To forward query string parameters from the resource request to the pagelet, set the optional parameter pt:forwardparams to ’true’ as shown in the example below.

```
<pt:ensemble.inject pt:name="lib:pagelet" pt:forwardparams="true"/>
```

**Note:** Some internal Oracle WebCenter Ensemble parameters are not forwarded, including SSO cookes and login redirects. Query string parameters that attempt to override any registered pagelet parameters are not forwarded.

When a pagelet request results in a 403, 404 or any other error code, Oracle WebCenter Ensemble can forward the error code and the error page itself to the browser for display to the user. To enable this option, set the attribute pt:onhttperror to one of the following values:

- comment (default)
- inline
- fullpage

For an example of using this code, seeSection 2.1.1, "Creating a Custom Pagelet with the Java Oracle WebCenter Interaction Development Kit (IDK) Proxy API" or Section 2.1.2, "Creating a Custom Pagelet with the .NET Oracle WebCenter Interaction Development Kit (IDK) Proxy API".You can also insert pagelets into non-proxied pages; for details, see Section 2.6.4, "Inserting Pagelets into Non-Proxied Pages".

**2.2.2.4.2 Accessing Authentication Data Using Oracle WebCenter Ensemble Adaptive Tags** The `pt:ensemble.authsourcedata` tag stores a list of available authentication sources for the currently requested resource in memory.

The data is stored as a collection, and each item in the collection is a data object containing information about the authentication source (prefix, name, description) accessible through the data object dot notation (`$curauth.name`).

```
<pt:ensemble.authsourcedata pt:key="sources"/>
<pt:logic.foreach pt:data="sources" pt:var="source">
<pt:logic.value pt:value="$source.prefix"/>
<pt:logic.value pt:value="$source.name"/>
<pt:logic.value pt:value="$source.description"/>
<pt:logic.separator><br>-----<br></pt:logic.separator>
</pt:logic.foreach>
```

This example uses logic tags to display information about each authentication source. For details on logic tags, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)".

**2.2.2.4.3 Accessing the Login URL Using Oracle WebCenter Ensemble Adaptive Tags** The `pt:ensemble.loginlink` tag stores the URL prefix for the login page in memory using the given key and scope.

The login prefix will end with a forward slash. This login prefix should be followed by the page suffix for the page that should be displayed after login. For example, if the external URL prefix of the resource is http://www.ensemble.com/app/ and the desired page after login is http://www.ensemble.com/app/pages/mainpage.html, then the full login link would be made by adding pages/mainpage.html to the login link prefix.

```
<pt:ensemble.loginlink pt:level="4" pt:key="loginurlprefix"/>
var loginLink = "<pt:logic.value pt:value="$loginurlprefix"/>" +
"pages/mainpage.html";
```

**2.2.2.4.4 Accessing Resource Data Using Oracle WebCenter Ensemble Adaptive Tags** The `pt:ensemble.resourcedata` tag stores data for a specific Oracle WebCenter Ensemble resource, if available, in memory as a data object.

The data object contains information about the resource (name, description, urlprefix, secureurlprefix) accessible through the data object dot notation (`$resource.name`). If the resource does not have a description, urlprefix, or secureurlprefix, the data will not be available.

```
<pt:ensemble.resourcedata pt:name="Welcome Resource" pt:key="resource"/>
```

```
<pt:logic.value pt:value="$resource.name"/>
<pt:logic.value pt:value="$resource.description"/>
<pt:logic.value pt:value="$resource.urlprefix"/>
<pt:logic.value pt:value="$resource.secureurlprefix"/>
```

**2.2.2.4.5 Accessing User Roles Using Oracle WebCenter Ensemble Adaptive Tags** The `role*` tags in the Oracle WebCenter Ensemble tag library allow pagelets to modify content based on the role of the current user.

For details on roles, see Section 2.6.5, "About Oracle WebCenter Ensemble Security".

The `pt:ensemble.roleexpr` tag evaluates a role expression and stores the result as a boolean in memory. This tag is designed to work with the `pt:logic.if` tag as shown below.

```
<pt:ensemble.roleexpr pt:expr="hasRole Default Role from Seed State"
pt:key="boolvalue"/>
<pt:logic.if pt:expr="$boolvalue">
<pt:logic.iftrue>
  <!-- This is displayed if expr evaluates to true. -->
</pt:logic.iftrue>
<pt:logic.iffalse>
  <!-- This is displayed if expr evaluates to false. -->
</pt:logic.iffalse>
</pt:logic.if>
```

The `pt:ensemble.rolelist` tag stores a list of the roles for the current user in memory. The data is stored as a collection, and each item in the collection is a variable containing the role name. This can be used with the `logic.foreach` tag to iterate over role data as shown below.

```
<pt:ensemble.rolelist pt:key="roles"/>
<pt:logic.foreach pt:data="roles" pt:var="role">
<pt:logic.value pt:value="$role"/>
<pt:logic.separator></pt:logic.separator>
</pt:logic.foreach>
```

For details on logic tags, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)".

### 2.2.2.5 Common Adaptive Tag Library (pt:common)

The Common tag library (pt:common) provides access to useful functionality, including URL transformation and namespace definition. This library also allows you to insert error information in the page, and CSS and JavaScript references in the Head element in a proxied HTML page.

The Common tag library is a cross-platform tag library that can be used in both Oracle WebCenter Interaction and Oracle WebCenter Ensemble.

For a full list of tags and attributes, see the tagdocs.

***Table 2–3 Tags in the Common Adaptive Tag Library***

| Tag | Function | More Information |
|---|---|---|
| `pt:common.namespace` | Defines a token for use in JavaScript functions and HTML elements to ensure unique names in an aggregated page. | Section 2.2.2.5.3, "Defining a Unique Namespace Token Using Adaptive Tags" |
| `pt:common.url` | Transforms URLs that should be proxied. | Section 2.2.2.5.5, "Transforming URLs Using Adaptive Tags" |
| `pt:common.transformer` | Disables and enables transformation on a proxied page. | Section 2.2.2.5.5, "Transforming URLs Using Adaptive Tags" |

*Table 2–3 (Cont.) Tags in the Common Adaptive Tag Library*

| Tag | Function | More Information |
|---|---|---|
| `pt:common.error` | Displays errors on the page so that they can be placed and formatted as desired. | Section 2.2.2.5.4, "Displaying Errors Using Adaptive Tags" |
| `pt:common.errorcode` | Stores a collection of the current error codes in memory. | Section 2.2.2.5.4, "Displaying Errors Using Adaptive Tags" |
| `pt:common.errortext` | Displays the current error text on the page so that it can be placed and formatted as desired. Only the first error message will be displayed. Used as singleton only (does not display the contents of the tag). | Section 2.2.2.5.4, "Displaying Errors Using Adaptive Tags" |
| `pt:common.headincludes` | Allows JavaScript and Style Sheet include information to be added to a specific point in the Head element of an HTML page, as required by the XHTML specification. | Section 2.2.2.5.2, "Adding Header Content Using Adaptive Tags" |
| `pt:common.includeinhead` | Marks JavaScript and CSS information to be included in the Head element of the HTML page by the pt:common.headincludes tag. | Section 2.2.2.5.2, "Adding Header Content Using Adaptive Tags" |
| `pt:common.userinfo` | Displays a specific user information setting. | Section 2.2.2.5.1, "Accessing User Information Using Adaptive Tags" |

**2.2.2.5.1 Accessing User Information Using Adaptive Tags** You can use the `pt:common.userinfo` tag to access specific user information settings.

The `pt:common.userinfo` tag is replaced with the value of the User Information setting specified in the `pt:info` attribute. The name attribute is case sensitive.

```
<pt:common.userinfo pt:info="FullName"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
```

**Note:** You must configure the Resource to send the appropriate user information settings to the pagelet.

**2.2.2.5.2 Adding Header Content Using Adaptive Tags** The `pt:common.includeinhead` and `headincludes` tags allow you to include custom JavaScript and CSS information in the Head element of the HTML page.

The `pt:common.includeinhead` tag marks the JavaScript and CSS information to be included in the Head element of the HTML page by the `pt:common.headincludes` tag. If a .js or .css file is marked for inclusion multiple times, it will only be included once. JavaScript generated by tags will also be included.

**Note:** This tag will be ignored during automatic in-place refresh requests. Custom in-place refresh solutions must ensure that JavaScript gets included correctly.

```
<pt:common.includeinhead>
<script type="text/javascript"><!-- JavaScript --></script>
<script type="text/javascript" src="http://test.com/test.js"></script>
<link type="text/css" rel="stylesheet" href="http://test.com/test.css"></link>
</pt:common.includeinhead>
```

The `pt:common.headincludes` tag adds JavaScript and stylesheet include information defined by the `pt:common.includeinhead` tag to the Head element of the HTML page, as required by the XHTML specification. If no `pt:common.headincludes` tag is present, JavaScript will be included at the bottom of the Head element, and a Head element will be inserted if one does not exist.

```
<head>
<script type="text/javascript" src="http://test.com/main.js"></script>
</head>
```

#### 2.2.2.5.3 Defining a Unique Namespace Token Using Adaptive Tags

It is an established best practice to include the pagelet ID in the name of any JavaScript functions and HTML elements to ensure unique names when the code is combined with markup from other pagelets on an aggregated page.

The `pt:common.namespace` tag allows you to define your own token, which is replaced with the pagelet ID. The token must follow these specifications:

- Valid values for the token must be in the ASCII range 0x21 to 0x7E, excluding "<" (0x3C).
- The scope of the token runs from the tag defining it to the end of the file; you cannot use a token prior to defining it.
- A second pt:namespace tag with a different token redefines it; two tokens cannot be defined at the same time.

```
<pt:common.namespace pt:token="$$TOKEN$$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
<a onclick="doStuff$$TOKEN$$();" href="#">do stuff</a>
<script>
function doStuff$$TOKEN$$() {
alert("hello");
}
</script>
```

#### 2.2.2.5.4 Displaying Errors Using Adaptive Tags

The error* tags in the Common library allow you to insert and format error messages within the page that contains the tag(s).

The `pt:common.error` tag displays errors on the page, placed and formatted as desired. If the `pt:common.errortext` tag is included inside this tag, the contents of the tag will only be processed if there is an error. If the child tag is not present, any error messages will be formatted and displayed from this tag in the standard style.If the `pt:common.errortext` tag is included, only the first error message will be displayed. Other errors, as well as exception stack traces and extended error messages, will be ignored. The `pt:common.errorcodes` tag stores a collection of the current error codes in memory. If the error has already been displayed, no error codes will be available. These error codes can be accessed using the `pt:logic.foreach` tag as shown below.

> **Note:** If these tags are displayed on a page, errors will no longer be displayed in the normal error location and will not be available after the tag has been displayed.

```
<pt:common.errorcode pt:key="errorcodes"/>
<pt:logic.foreach pt:data="errorcodes" pt:var="code">
<pt:common.errortext/>
```

**2.2.2.5.5 Transforming URLs Using Adaptive Tags** The `pt:common.url` and `pt:common.transformer` tags allow you to create and manipulate proxiedURLs.

The `pt:common.url` tag is used to transform URLs that should be proxied. If the URL in the `pt:href` attribute is outside the proxied Resource, it will be transformed to an absolute URL. This feature does not generate a link in HTML; it obtains the URL as a string and passes it to a client-side function, as shown in the following example.

```
<script>
function myFunction()
{
document.write("<pt:common.url pt:href="myURL"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>");
}
```

The `pt:common.transformer` tag allows you to turn off JavaScript URL transformation in a proxied page. Set the `pt:fixurl` attribute to "off" as shown below.

```
<pt:common.transformer pt:fixurl="off"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
```

The transformer will not insert calls to the JavaScript URL transformation function for the rest of the file, unless you switch the feature back on in a subsequent directive (with a `pt:fixurl` attribute of "on").

### 2.2.2.6 Logic Adaptive Tag Library (pt:logic)

Logic tags handle basic logic, including creating data objects and collections, setting shared variables, evaluating expressions, and looping over a data collection.

The pt:logic tag library is a cross-platform tag library that can be used in both Oracle WebCenter Interaction and Oracle WebCenter Ensemble.

> **Note:** Many logic tags have a `pt:scope` attribute. The valid scope values are: tag, portlet request (pagelet request), http request, session, persistent session, and application. The default is portlet request scope.

For a full list of tags and attributes, see the tagdocs.

*Table 2–4 Tags in the Logic Adaptive Tag Library*

| Tag | Function | More Information |
|---|---|---|
| `pt:logic.data` | Creates a data object (collection of name=value pairs) and stores it in a shared variable using the key supplied. | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |
| `pt:logic.concat` | Concatenates two values into one and sets the new value in a variable with a specified name. | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |
| `pt:logic.variable` | Stores a shared variable using the key and value supplied. Designed for use with attribute replacement or with the `pt:logic.value` tag. | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |
| `pt:logic.collection` | Creates a collection of data objects and stores it in a shared variable using the key supplied. | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |

*Table 2–4 (Cont.) Tags in the Logic Adaptive Tag Library*

| Tag | Function | More Information |
|---|---|---|
| `pt:logic.collectionlength` | Evaluates the length of a collection and stores the result in memory. | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |
| `pt:logic.value` | Evaluates an attribute and displays the referenced value. Used as singleton only (does not display the contents of the tag). | Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags" |
| `pt:logic.boolexpr` | Evaluates a boolean expression and stores the result as a boolean in memory. Designed to work with the `pt:logic.if` tag. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic.intexpr` | Evaluates an integer expression and stores the result as a boolean in memory. Designed to work with the `pt:logic.if` tag. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic.stringexpr` | Evaluates whether or not two strings are equal and stores the result as a boolean in memory. The case must match. Designed to work with the `pt:logic.if` tag. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic.containsexpr` | Checks if a collection contains a specific data element and sets a specified variable to true or false. Designed to work with the `pt:logic.if` tag. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic.if` | Evaluates an expression and displays either the `pt:logic.iftrue` or `pt:logic.iffalse` tag contents. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic:iffalse` | Displayed if the surrounding `pt:logic.if` tag evaluates to false. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic:iftrue` | Displayed if the surrounding `pt:logic.if` tag evaluates to true. | Section 2.2.2.6.1, "Evaluating Expressions Using Adaptive Tags" |
| `pt:logic.foreach` | Allows looping over a data collection. Supports tag and portlet request scope only. | Section 2.2.2.6.2, "Looping Over Data Collections Using Adaptive Tags" |
| `pt:logic.separator` | Inserts a separator between the elements of a for each loop. | Section 2.2.2.6.2, "Looping Over Data Collections Using Adaptive Tags" |

#### 2.2.2.6.1 Evaluating Expressions Using Adaptive Tags

The `pt:logic.boolexpr`, `intexpr`, `stringexpr` and `containsexpr` tags work with the `pt:logic.if` tag to evaluate a range of expressions.

The sample code below determines whether the current value for the variable "title" is set to "Administrator". Variables can be set using the `pt:logic.data` or `pt:logic.variable` tags.

```
<pt:logic.stringexpr pt:expr="($title) == Administrator" pt:key="boolvalue"/>
<pt:logic.if pt:expr="$boolvalue">
<pt:logic.iftrue>
This is displayed if expr evaluates to true.
</pt:logic.iftrue>
<pt:logic.iffalse>
This is displayed if expr evaluates to false.
</pt:logic.iffalse>
</pt:logic.if>
```

For details on using shared variables, see Section 2.2.2.6.3, "Using Shared Variables in Adaptive Tags".

#### 2.2.2.6.2 Looping Over Data Collections Using Adaptive Tags

The `pt:logic.foreach` tag allows you to loop over collections of data.

The sample code below creates a table to store links for a navigation menu.

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<table cellpadding="5" cellspacing="0" width="100%" border="0">
<!-- loop starts here -->
<pt:logic.foreach pt:data="directorymenu" pt:var="temp">
<tr>
<td height="25" colspan="3" class="navSidebarText">
<pt:core.html pt:tag="img" src="$temp.img" alt="" border="0" align="absmiddle"
height="20" width="20" />
<pt:core.html pt:tag="a" href="$temp.url">
<pt:logic.value pt:value="$temp.title" />
</pt:core.html>
</td>
</tr>
</pt:logic.foreach>
</table>
</span>
```

This table can then be populated with links.

#### 2.2.2.6.3 Using Shared Variables in Adaptive Tags

The `pt:logic.data`, `variable`, and `collection` tags allow you to store editable shared variables, which can be used in attribute value replacement or with the `pt:logic.value` tag.

The `pt:logic.data` tag stores a data object (a name=value pair) as an editable shared variable using the key passed in. The `pt:logic.variable` tag stores an editable shared variable using the key and value passed in. If either tag is used inside the `pt:logic.collection` tag, the variables are stored directly in the parent collection. If the tag is used alone, the key attribute is required. The variable is only stored after the tag is finished processing all its contents. A collection can only contain a single type of variable, such as string variables or data objects.

---

**Note:** If a variable or collection with the same name already exists, it will be overwritten. If the preexisting variable is not editable, the tag will fail. Variable names cannot contain the reserved character '.'.

---

```
<pt:logic.variable pt:key="title" pt:value="Administrator"/>

<pt:logic.data pt:key="myurl" name="Home" url="http://edocs.bea.com"/>

<pt:logic.collection pt:key="testcollection">
<pt:logic.data url="http://www.myco.com" name="My company"/>
<pt:logic.data url="http://www.otherco.com" name="Other company"/>
</pt:logic.collection>

<pt:logic.collection pt:key="teststringcollection">
<pt:logic.variable pt:value="my string data"/>
<pt:logic.variable pt:value="my other string data"/>
</pt:logic.collection>
```

The `pt:logic.value` tag displays the value of the variable referenced by the `pt:value` attribute. Variables can be set using the `pt:logic.data` or `pt:logic.variable` tags as explained in the previous section. This tag can be used to reference localized strings in message files.

```
<pt:logic.value pt:value="$title"/>
<pt:logic.value pt:value="$myurl.Home"/>
```

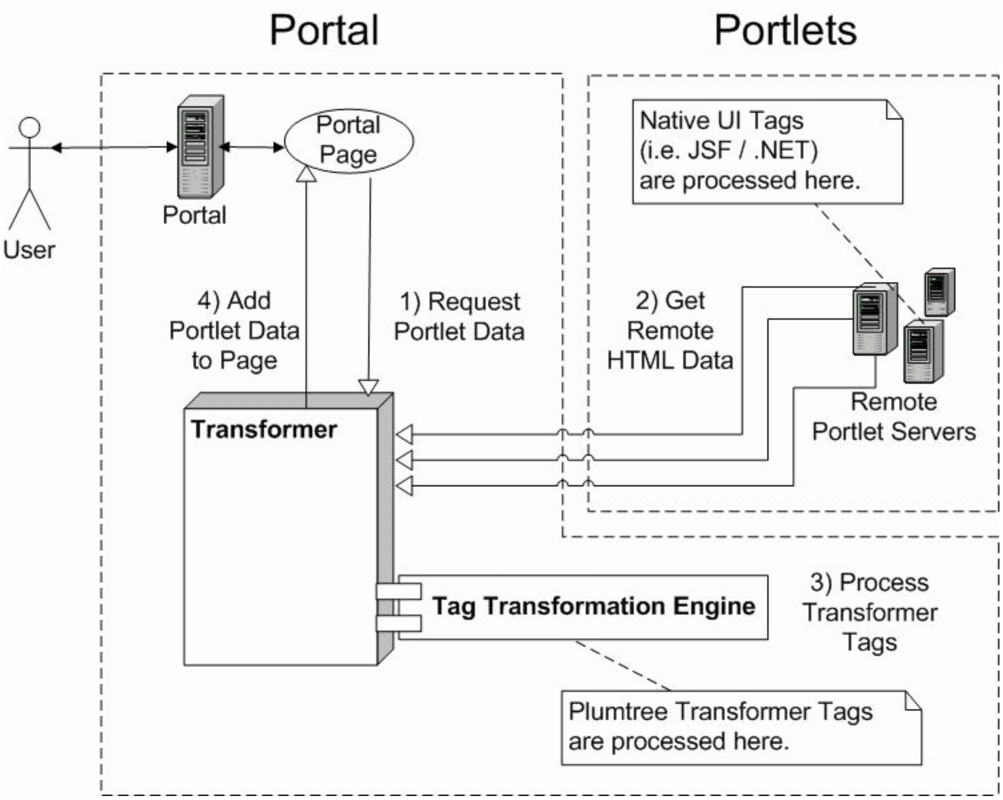
For details on referencing localized strings using tags, see Section 2.2.2.2, "Using Internationalized Strings in Adaptive Tags".

#### 2.2.2.7 About Adaptive Tag Control Flow

This page describes the control flow of a typical request that makes use of adaptive tags.

1. First, the proxied page requests pagelet data from the transformer.
2. The transformer retrieves the requested pagelets from the external resources. Native UI tags, such as JSP Tags or .NET Web controls, are processed on the external resource before the HTML is returned to the transformer.
3. The transformer converts the HTML into markup data including both HTML and adaptive tags. This markup data is passed to the Tag Transformation Engine, which processes the tags and converts them into standard HTML.
4. Finally, the HTML is returned to the page where it is displayed to the end user.

***Figure 2–1 Tag Control Flow***

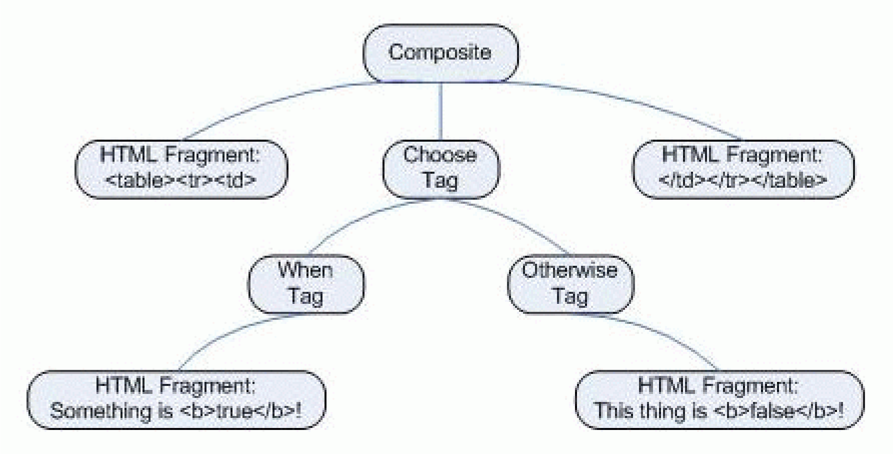

Portal
Portlets
User
Portal
Portal Page
Native UI Tags (i.e. JSF / .NET) are processed here.
4) Add Portlet Data to Page
1) Request Portlet Data
2) Get Remote HTML Data
Remote Portlet Servers
Transformer
Tag Transformation Engine
3) Process Transformer Tags
Plumtree Transformer Tags are processed here.

The **Tag Transformation Engine** converts markup data from the transformer into a tree of HTML and adaptive tags. The Engine moves through the tree and outputs

HTML and processes the tags. When a tag is processed, it can cause all of its child nodes to be processed, or it can skip that entire section of the tree.

The figure below shows an example of a tree. In this example, when the `choose` tag is executed, it determines whether or not the current user matches the conditions in the choose clause. If it does, the `when` tag will display the HTML inside the tag. If not, the `otherwise` tag will display its HTML

***Figure 2–2 Tag Transformation Engine Tree***


Composite
HTML Fragment: <table><tr><td>
Choose Tag
HTML Fragment: </td></tr></table>
When Tag
Otherwise Tag
HTML Fragment: Something is <b>true</b>!
HTML Fragment: This thing is <b>false</b>!

For details on these tags, see Section 2.2.2.6, "Logic Adaptive Tag Library (pt:logic)".

### 2.2.2.8 Creating Custom Adaptive Tags

The Adaptive Tag Framework allows you to create custom tags for use in pagelets and proxied pages.

The `ATag` class is the base class used to write custom tags. To implement a custom tag, follow the steps below.

1. To implement a new tag, you must have a tag library. A tag library is simply a .jar or .dll file with exactly one class that implements `ITagLibraryMetaData`.

   **Java**

   ```
   public static final TagLibraryMetaData LIBRARY = new TagLibraryMetaData
   ("Sample Tags", "sample", "This library provides sample tags.", 1.0);
   ```

   **.NET**

   ```
   public static readonly TagLibraryMetaData LIBRARY = new TagLibraryMetaData
   ("Sample Tags",  "sample", "This library provides sample tags.", 1.0);
   ```

2. Create one public static final `ITagMetaData` member variable that provides the name and description of the tag. Create a public static final `RequiredTagAttribute` or `OptionalTagAttribute` member variable for every attribute that the tag supports. You can also use standard HTML and XML attributes; see Section 2.2.2.3, "Using Variables in Adaptive Tags".

   **Java**

   ```
   public static final ITagMetaData TAG;
   public static final RequiredTagAttribute MESSAGEATTRIBUTE;
   public static final OptionalTagAttribute LOCATIONATTRIBUTE;
   ```

```
static
{
TAG = new TagMetaData("hellolocation", "This tag displays a hello message for
the given location.");
MESSAGEATTRIBUTE = new RequiredTagAttribute( "message", "The message to display
for hellolocation tag", AttributeType.STRING);
LOCATIONATTRIBUTE = new OptionalTagAttribute("location", "The sample location
attribute for hellolocation tag", AttributeType.STRING, "World");
}
```

**.NET**

```
public static readonly ITagMetaData TAG;
public static readonly RequiredTagAttribute MESSAGEATTRIBUTE;
public static readonly OptionalTagAttribute LOCATIONATTRIBUTE;

static HelloLocationTag()
{
TAG = new TagMetaData("hellolocation", "This tag displays a hello message for
the given location.");
MESSAGEATTRIBUTE = new RequiredTagAttribute( "message", "The message to display
for hellolocation tag", AttributeType.STRING);
LOCATIONATTRIBUTE = new OptionalTagAttribute("location", "The sample location
attribute for hellolocation tag", AttributeType.STRING, "World");
}
```

Type validation is performed by the tag framework automatically. If an optional attribute is not present in the HTML, the tag framework will use the default value. In the same code below, the optional attribute has a default value of "World.".

3. Implement the `DisplayTag` abstract method. Use this method to create and display HTML. To display any HTML and tags defined within the tag, call `ProcessTagBody` and return the resulting HTML. The sample code below adds the "Hello" string with a user-specified location to an `HTMLElement` and returns it to be displayed.

   **Java**

   ```
   public HTMLElement DisplayTag()
   {
   String strLocation = GetTagAttributeAsString(LOCATIONATTRIBUTE);
   String strMessage = GetTagAttributeAsString(MESSAGEATTRIBUTE);
   HTMLElementCollection result = new HTMLElementCollection();
   result.AddInnerHTMLString(strMessage + strLocation + "!");
   return result;
   }
   ```

   **.NET**

   ```
   public override HTMLElement DisplayTag()
   {
   String strLocation = GetTagAttributeAsString(LOCATIONATTRIBUTE);
   String strMessage = GetTagAttributeAsString(MESSAGEATTRIBUTE);
   HTMLElementCollection result = new HTMLElementCollection();
   result.ddInnerHTMLString(strMessage + strLocation + "!");
   return result;
   }
   ```

4. If the tag should not display any HTML contained within the tag, use the `GetTagType` method to return TagType.NO_BODY.

   **Java**

```
public TagType GetTagType()
{
return TagType.NO_BODY;
}
```

**.NET**

```
public override TagType GetTagType()
{
return TagType.NO_BODY;
}
```

5. Implement the `Create` abstract method to return a new instance of the tag.

   **Java**

   ```
   public ATag Create()
   {
   return new HelloLocationTag();
   }
   ```

   **.NET**

   ```
   public override ATag Create()
   {
   return new HelloLocationTag();
   }
   ```

The `ATag` class allows you to include a wide range of functionality in custom tags. For a full list of interfaces and methods, see the tagdocs. For details on deploying your custom tag, see Section 2.2.2.8.7, "Deploying Custom Adaptive Tags".

**2.2.2.8.1 Accessing Browser Session Information in Custom Adaptive Tags** To access browser session information from a custom adaptive tag, use the `IEnvironment` class.

The `IEnvironment` class provides access to information about the current request and user, including the following:

- **HTTP Request and Response**: Retrieve the Request or Response objects, or the Request URL. For example: `IXPRequest request = GetEnvironment().GetCurrentHTTPRequest();`
- **User information**: Retrieve the user's session, or key information including language, locale, time zone, and access style (standard, 508, or low bandwidth). For example: `String strTZ = GetEnvironment().GetTimeZone();`
- **VarPacks**: Retrieve any VarPacks associated with the application in which the tag is executed.

**2.2.2.8.2 Accessing Attributes in Custom Adaptive Tags** To access attributes used in a custom tag, use one of the `GetTagAttribute*` methods.

All basic data types are supported as attributes (defined in the `AttributeType` class), including boolean, char, double, int, long and string. The "pt:" attributes specify the logic for the tag, while any non-pt attributes specify the behavior of the resulting HTML tag. Non-pt attributes are only applicable in tags that output a simple HTML tag.

- To access pt attributes, use the appropriate `GetTagAttributeAs*` method using the attribute name. A method is provided for each supported attribute type, e.g., `GetTagAttributeAsLong`. The `GetTagAttribute` method is provided for backwards compatibility and should not be used.

1. First, define the attribute: `MODE = new OptionalTagAttribute("mode", "Turns debug mode on and off.", AttributeType.BOOLEAN, "true");`
2. Then, access the attribute in the `DisplayTag` method:`boolean bNewDebugMode = GetTagAttributeAsBoolean(MODE);`

- To access non-pt (XML/HTML) attributes, use the `GetXMLTagAttribute` method using the attribute name, or `GetXMLTagAttributesAsString` to retrieve all non-pt attributes. `result.AddInnerHTMLElement(new HTMLGenericElement("<a href=\"" + GetHREF() + "\" " + GetXMLTagAttributesAsString() + ">"));`

The `ITagMetaData`, `RequiredTagAttribute`, and `OptionalTagAttribute` objects pre-process tag attributes (presence, correct type, and default values). If the required attributes are not correct, an error is logged and the tag and its children are skipped. An HTML comment describing the tag and error is displayed instead.

**2.2.2.8.3 Storing and Accessing Custom Data in Custom Adaptive Tags** To store custom data as member variables using a custom tag, use the `SetStateVariable` or `SetStateSharedVariable` methods. To retrieve it, use `GetStateVariable` or `GetStateSharedVariable`.

Standard variables (stored with `SetStateVariable`) can only be accessed by tags in the same library. Shared variables (stored with `SetStateSharedVariable`) can be accessed by tags from any library. To prevent tags from other libraries from editing a shared variable, set `bOwnerEditOnly` to true when the shared variable is stored (tags in other libraries will still be able to read the variable).The `Scope` parameter determines who can see the data and how long it stays in memory. The following options are defined in the `Scope` class:

*Table 2–5 Options Defined in Scope Class*

| Option | Description |
|---|---|
| Application Scope | Data is visible to all tags and all users, and is only removed when the application is restarted. Therefore, care should be used when storing data on the application to make sure it does not become cluttered with large amounts of data. |
| HTTP Request Scope | Data will be visible to all tags in the same HTTP Request as the current tag, and is removed from memory when the HTTP Request is finished. |
| Session Scope | Data is visible to all tags for the current user, and is cleared from memory when a user logs out and logs in again. |
| Persistent Session Scope | Data is visible to all tags in the same HTTP session, and is only removed from memory when the browser is closed or the browser session times out. Note: Data is not cleared on user logout, so do not cache anything on this scope that could be considered a security risk if it was leaked to another user. Most tags should use Session Scope for HTTP Session data storage (as described above). |
| Portlet Request Scope | Data is visible to all tags in the same pagelet as the current tag, and is removed from memory when the pagelet is finished displaying. Tags in other pagelets on the same page will not be able to see the data. |
| Tag Scope | Data can only be seen by children of the current tag and is removed from memory when the tag is finished. (For example, in the following tags: `<pt:atag><pt:btag/></pt:atag><pt:ctag/>`, data stored in Tag Scope by "atag" would be visible to "btag" but not to "ctag.") |

If data is stored directly in the tag in member variables (not recommended), override the `ReleaseTag` method to release the data stored on the tag.

```
/**
```

```
* @see com.plumtree.portaluiinfrastructure.tags.ATag#ReleaseTag()
*/
public void ReleaseTag()
{
// Release all member variables.
m_strPreviousRequestURL = null;
}
```

---

**Note:** Displaying an `HTMLElement` in a tag and caching it so another tag can add more HTML is not supported. `HTMLElement` trees can be generated and stored for later use as long as they are self-contained trees and used in a read-only way. It is safest to clone a cached `HTMLElement` tree before trying to display it again to make sure there are no threading problems.

---

**Note:** It is a best practice not to use static fields for data storage in tags. Each tag instance is guaranteed to be accessed by only a single thread at a time, but there may be multiple threads accessing different instances of the same tag class at the same time, either from the same user or a different user. This means that any static fields must be accessed using synchronized methods. Since there can be multiple instances of the same tag running at the same time, state variables set in shared scopes (Session, Persistent Session and Application) could change values during the execution of a single tag.

---

**2.2.2.8.4 Including JavaScript in Custom Adaptive Tags** To include JavaScript in a tag, use the `AddJavaScript` method inside the `DisplayTag` method.

For example:

```
HTMLScriptCollection scriptCollection = new HTMLScriptCollection();
HTMLScript script = new HTMLScript("text/javascript");
scriptCollection.AddInnerHTMLElement(script);
script.AddInnerHTMLString("function myTest() { alert('test'); }");
AddJavaScript(scriptCollection);
```

To include common JavaScript that can be shared between multiple instances of a tag (i.e. JavaScript that is displayed once per page, regardless of how many tags of a certain type there are), override the `DisplaySharedJavaScript` method. `DisplaySharedJavaScript` is called automatically by the framework.

```
/**
* Adds the PTIncluder object to the client.  This object is used for
* retrieving JSComponent client classes from a page.
*/
public HTMLScriptCollection DisplaySharedJavaScript()
{
HTMLScriptCollection result = new HTMLScriptCollection();
HTMLScript script = new HTMLScript("text/javascript");
result.AddInnerHTMLElement(script); script.SetSrc("/myjsfile.js");
return result;
}
```

If there are errors in the tag and the JavaScript cannot be displayed properly, the tag should throw an `XPException` with an error message, and the tag framework will log the error and add the message and stack trace to the HTML as an HTML comment. The message contents will be HTML encoded before being added to the comment.

---

**Note:** JavaScript is not displayed in 508 mode for either method, since section 508 compliant browsers do not support JavaScript.

---

**2.2.2.8.5 Using Nested Tags in Custom Adaptive Tags** Tags can be used within other tags. To implement nested tags, use the `RequiredParentTag`, `RequiredChildTag` and `RelatedChildTag` member variables.

The outer tag is referred to as the "parent" tag. Any tags within a parent tag are referred to as "child" tags of that tag. If the tag is only intended for use within a particular parent tag, create a public static final `RequiredParentTag` member variable. If there are multiple `RequiredParentTag` members, at least one of the parent tags must be present for the child tag to function. If the tag must include a particular child tag to function, create a public static final `RequiredChildTag` member variable for each tag that is required inside the parent tag. If the child tag is not required for the parent tag to function, but is still related to that tag, create a public static final `RelatedChildTag` member variable instead.

```
public static final RequiredChildTag DATA_OBJECT;
static
{
... DATA_OBJECT = new RequiredChildTag(DataObjectTag.TAG);
}
```

---

**Note:** If required parent or child tags are missing when a tag is displayed, the tag framework will not process the incorrect tag and will add an error message to the HTML as an HTML comment.

---

**2.2.2.8.6 Implementing Non-Standard Custom Adaptive Tag Types** To implement non-standard tag types in custom adaptive tags, including 508-accessible, looping or singleton tags, override the associated method.

- To display a custom tag in non-standard access styles (508 or low bandwidth), override the `SupportsAccessStyle` method. The default implementation of the `SupportsAccessStyle` method will cause the tag to be skipped in 508 and low-bandwidth mode. Make sure that tags that support 508 mode can function without JavaScript, since JavaScript will not be displayed in 508 mode.
- If the tag displays the tag body more than once (looping tag), override the `GetTagType()` method and return TagType.LOOPING.
- If the tag never displays the tag body (singleton tag), override `GetTagType()` and return TagType.NO_BODY.

**2.2.2.8.7 Deploying Custom Adaptive Tags** To deploy custom adaptive tags, follow these steps.

1. Navigate to PORTAL_HOME\settings\portal and open CustomTags.xml in a text editor (you might need to make the file writable).
2. Find the `<AppLibFiles>` tag and add a new entry using the name of the .jar/.dll file used to define the custom tag library (e.g., mytags).

```
<AppLibFiles>
<libfile name="sampletags"/>
</AppLibFiles>
```

3. Add the custom implementation (.jar/.dll) to Oracle WebCenter Ensemble hierarchy:
   - Java: Copy the custom .jar file to ENSEMBLE_HOME\lib\java and add it to the .war file in ENSEMBLE_HOME\webapp. (You must stop Oracle WebCenter Ensemble while modifying portal.war because it will be locked while Oracle WebCenter Ensemble is running.)
   - .NET: Copy the custom .dll file to ENSEMBLE_HOME\webapp\portal\bin.
4. Run a clean build of Oracle WebCenter Ensemble to refresh all the jar files associated with Oracle WebCenter Ensemble.
5. Once you have deployed your code, create a pagelet that contains the tag. Custom adaptive tags must either include the correct XML namespace or be contained within another tag that does. The simplest way is to put the HTML inside a span. Custom adaptive tags must use the pt:libraryname.tagname and pt:attributename format. The sample code below references the custom tag from Section 2.2.2.8, "Creating Custom Adaptive Tags".

```
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:sample.hellolocation pt:message="Hello" pt:location="San Francisco"/>
</span>
```

6. Add the pagelet to a page and view the page. Test all custom tags.

## 2.2.3 Oracle WebCenter Interaction Scripting Framework

The Oracle WebCenter Interaction Scripting Framework is a client-side JavaScript library that provides services to pagelets and proxied pages. The Portlet Communication Component (PCC) is contained within the Scripting Framework.

The Oracle WebCenter Interaction Scripting Framework allows pagelets to:

- **Store and share session state through browser level variables.** Browser-level variables can be stored and shared among pagelets, even if they are not on the same page. For example, a value entered by the user in one pagelet can be retrieved by another. The Scripting Framework acts as an intermediary, allowing all pagelets access to all values stored in a common session. For details, see Section 2.3, "Session Preferences".
- **Leverage page-level events.** A pagelet can respond when specific events happen, such as when the page loads or when the browser focus changes. For details, see Section 2.2.3.2, "Using Oracle WebCenter Interaction Scripting Framework Event Notification".
- **Refresh pagelet content without reloading the page.** pagelets can reload their internal content without refreshing the page. For details, see Section 2.2.3.3, "Using In-Place Refresh".

For a full list of classes and methods, see the JSPortlet API documentation.

### 2.2.3.1 Oracle WebCenter Interaction Scripting Framework Development Tips

These tips and best practices apply to all code that utilizes the Oracle WebCenter Interaction Scripting Framework.

- **Use unique names for all forms and functions.** Use the GUID of a pagelet to form unique names and values to avoid name collisions with other code on the page. You can append the pagelet ID using the `pt:namespace` and `pt:token` tags, as shown in the code below.

```
<pt:namespace pt:token="$$TOKEN$$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>
<a onclick="doStuff$$TOKEN$$();" href="#">do stuffa
onclick="doStuff$$TOKEN$$();" href="#">do stuff</a>
<script>
function doStuff$$TOKEN$$() {
alert("hello");
}
</script>
```

Valid values for the token are in the ASCII range 0x21 to 0x7E, excluding "<" (0x3C). The scope of the token runs from the tag defining it to the end of the file; you cannot use a token prior to defining it. A second `pt:namespace` tag with a different token redefines it; two tokens cannot be defined at the same time.

- **Proxy all URLs.** You cannot make a request to a URL whose host/port differs from that of the calling page. All URLs requested through JavaScript must be proxied. For details, see Section 1.3, "About Server Communication and the Proxy".
- **Check for Scripting Framework support.** It is good practice to include code that determines whether or not the component is present. Ideally, your pagelet should be able to handle either situation. The simplest solution is to precede your code with an If statement that alerts the user if the Scripting Framework is not supported.

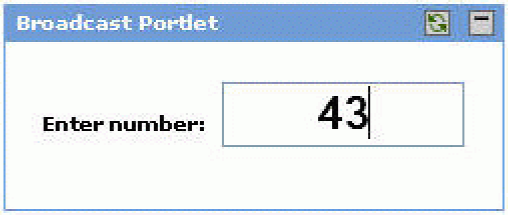
```
<script>
if (PTPortlet == null)
  {
  if (document.PCC == null)
    {
    alert("This portlet only works in portals that support the JSPortlet API or
Portlet
    Communication Component (PCC). The portlet will be displayed with severely
reduced
    functionality. Contact your Administrator.");
    }
  }
else
  {
  [scripting code here]
  }
</script>
```

- **Close all popup windows opened by a pagelet when the window closes.** The Scripting Framework can be used to close popup windows using the `onunload` event.
- **Do not assume that browsers will process script blocks/includes added through the innerHTML property.** Add all required JavaScript to the page in advance:
  - Microsoft Internet Explorer: Add the defer attribute to the script tag.
  - Netscape: Use RegExp to parse the response and look for the script, then evaluate it.

#### 2.2.3.2 Using Oracle WebCenter Interaction Scripting Framework Event Notification

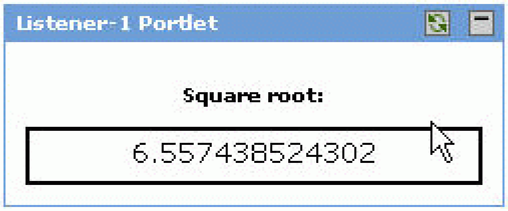
The Oracle WebCenter Interaction Scripting Framework allows pagelets to respond to both page-level events and custom events raised by other pagelets.

The `registerForWindowEvent` and `registerOnceForWindowEvent` methods in the Oracle WebCenter Interaction Scripting Framework provide pagelets with access to page-level events. For a complete list, see Section 2.2.3.2.1, "Page-Level Events for Use with the Oracle WebCenter Interaction Scripting Framework". To register for notification of these events, pass in the name of the event and the name of the method that should be called when it occurs. When a page-level event is raised, the JavaScript event object is passed to the event handler as an argument. The Oracle WebCenter Interaction Scripting Framework also allows pagelets to raise and respond to custom events using `raiseEvent` and `registerForEvent`. The Broadcast-Listener design pattern illustrates an important example of using notification services with session preferences. Users can select an item or perform some other action in a "broadcast" pagelet, which causes the content in other related "listener" pagelets to be redrawn. In the following example, the broadcast pagelet displays a form that allows you to enter a number in a text box.



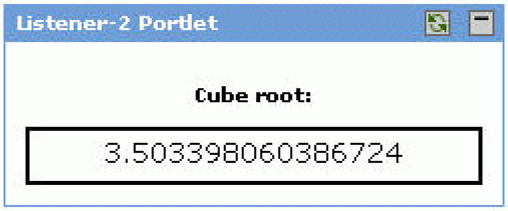
Enter number: 43

When the user enters a number in the text box, the values in the listener pagelets change. The first listener pagelet displays the square root of the number entered in the broadcast pagelet.



Square root: 6.557438524302

The second listener pagelet displays the cube root of the number entered in the broadcast pagelet.



Cube root: 3.503398060386724

The following steps summarize how the pagelets work:

- On load, each listener pagelet calls its own instance method (`registerForEvent`) to register for events of type 'onBroadcastUpdate'.
- On each onkeyup event that occurs in the "Enter number" text box, the broadcast pagelet sets a session preference to the value entered in the text box, and calls its own instance method (`raiseEvent`) to raise an event of type 'onBroadcastUpdate'.

- When the onBroadcastUpdate event is raised or the page is reloaded, each listener pagelet retrieves the session preference set by the broadcast pagelet and computes a new value to display based on the value of the preference.

### Broadcast Pagelet

```
<div style="padding:10px;" align="center">
<p><b>Enter number:</b>
&nbsp;<input type="text"
style="font-size:22px;font-weight:bold;text-align:center;"
id="broadcast_prefName" value="4" size="7" onkeyup="broadcast_
setPrefs(this.value)"></p>
<br>
</div>

<script type="text/javascript">

function broadcast_setPrefs(val)
{
    var prefName = 'broadcastNumber';
    var prefValue = val;
    PTPortlet.setSessionPref(prefName,prefValue);

    var broadcastPortlet =
PTPortlet.getPortletByGUID('{D9DFF3F4-EAE7-5478-0F4C-2DBD94444000}');

      if (!broadcastPortlet)
    {
        broadcast_debug('Could not locate PTPortlet object which corresponds to
<b>Broadcast Portlet</b> on page.');
        return;
        }

      broadcast_debug('<b>Broadcast Portlet</b> raising onBroadcastUpdate
event.');
    broadcastPortlet.raiseEvent('onBroadcastUpdate',false);

}

function broadcast_debug(str)
{
    if (window.PTDebugUtil)
    {
        PTDebugUtil.debug(str);
    }
}
</script>
```

### Listener Pagelet #1

```
<div style="padding:10px;" align="center">
<p><b>Square root:</b>
<div style="height:21px;border:2px solid
black;padding:2px;overflow:visible;font-size:14px;"id="listener1-swatch">
</div>
</div>

<script>

function listener1_update()
{
    var broadcastNumber = parseFloat(PTPortlet.getSessionPref('broadcastNumber'));
    if (isNaN(broadcastNumber))
    {
        listener1_error('<b>Listener-1 Portlet</b> cannot parse number from
session pref broadcastNumber');
        return;
    }

      listener1_debug('<b>Listener-1 Portlet</b> computing square root of ' +
broadcastNumber);
      var swatch = document.getElementById('listener1-swatch');
    swatch.innerHTML = Math.sqrt(broadcastNumber);
}

function listener1_debug(str)
{
    if (window.PTDebugUtil)
    {
        PTDebugUtil.debug(str);
    }
}

function listener1_error(str)
{
    if (window.PTDebugUtil)
    {
        PTDebugUtil.error(str);
    }
}

function listener1_getPortlet()
{
    var portletGUID = '{D9DFF3F4-EAE7-5478-0F4C-2DBDB4F4A000}';
    var listener1Portlet = PTPortlet.getPortletByGUID(portletGUID);
    return listener1Portlet;
}

var listener1Portlet = listener1_getPortlet();
if (listener1Portlet)
{
    listener1Portlet.registerForEvent('onBroadcastUpdate','listener1_update');
    listener1_debug('<b>Listener-1 Portlet</b> registered refreshOnEvent for event
onBroadcastUpdate');
    listener1Portlet.registerForEvent('onload','listener1_update');
}

</script>
```

### Listener Pagelet #2

```
<div style="padding:10px;" align="center">
<p><b>Cube root:</b>
<div style="height:21px;border:2px solid
black;padding:2px;overflow:visible;font-size:14px;"id="listener2-swatch">
</div>
</div>

<script>
var listener2_oneThird = (1/3);

function listener2_update()
{
    var broadcastNumber = parseFloat(PTPortlet.getSessionPref('broadcastNumber'));
    if (isNaN(broadcastNumber))
    {
        listener2_error('<b>Listener-2 Portlet</b> cannot parse number from 
session pref broadcastNumber');
        return;
    }

    listener2_debug('<b>Listener-2 Portlet</b> computing square root of ' + 
broadcastNumber);

    var swatch = document.getElementById('listener2-swatch');
    swatch.innerHTML = Math.pow(broadcastNumber,listener2_oneThird);
}

function listener2_debug(str)
{
    if (window.PTDebugUtil) 
    { 
        PTDebugUtil.debug(str); 
    }
}

function listener2_error(str)
{
    if (window.PTDebugUtil) 
    { 
        PTDebugUtil.error(str); 
    }
}

function listener2_getPortlet()
{
    var portletGUID = '{D9DFF3F4-EAE7-5478-0F4C-2DBDCA1C7000}';
    var listener2Portlet = PTPortlet.getPortletByGUID(portletGUID);
    return listener2Portlet;
}

var listener2Portlet = listener2_getPortlet();
if (listener2Portlet)
{
    listener2Portlet.registerForEvent('onBroadcastUpdate','listener2_update');
    listener2_debug('<b>Listener-2 Portlet</b> registered refreshOnEvent for event 
onBroadcastUpdate');
    listener2Portlet.registerForEvent('onload','listener2_update');
}

</script>
```

#### 2.2.3.2.1 Page-Level Events for Use with the Oracle WebCenter Interaction Scripting Framework

The Oracle WebCenter Interaction Scripting Framework automatically has access to the following page-level events.

*Table 2–6 Page-Level Events*

| Event | Triggered: |
|---|---|
| onload | immediately after the browser loads the page |
| onbeforeunload | prior to a page being unloaded (browser window closes or navigates to different location) |

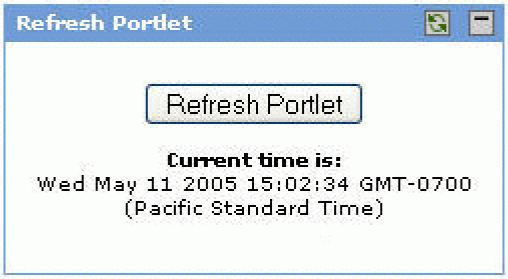
*Table 2–6 (Cont.) Page-Level Events*

| Event | Triggered: |
|---|---|
| onunload | immediately before the page is unloaded (browser window closes or navigates to different location) |
| onactivate | the page is set as the active element (receives focus) |
| onbeforeactivate | immediately before the page is set as the active element (receives focus) |
| ondeactivate | when the active element is changed from the current page to another page in the parent document |
| onfocus | when the page receives focus |
| onblur | when the page loses focus |
| oncontrolselect | when the user is about to make a control selection of the page |
| onresize | when the size of the page is about to change |
| onresizestart | when the user begins to change the dimensions of the page in a control selection |
| onresizeend | when the user finishes changing the dimensions of the page in a control selection |
| onhelp | when the user presses the F1 key while the browser is the active window |
| onerror | when an error occurs during page loading |
| onafterprint | immediately after an associated document prints or previews for printing |

### 2.2.3.3 Using In-Place Refresh

To refresh pagelet content in place, without affecting other content on the page, use the Oracle WebCenter Interaction Scripting Framework to implement in-place refresh.

Many pagelets display data that is time sensitive. In some cases, users should be able to navigate across links within a pagelet without changing or refreshing the rest of the page. You can refresh pagelet content on command, associate the refresh action with an event (`refreshOnEvent`), or program the pagelet to refresh at a set interval (`setRefreshInterval`). The Oracle WebCenter Interaction Scripting Framework also contains methods for expanding and collapsing pagelets. In the simplified example below, the refresh pagelet displays a "Refresh Portlet" button. Clicking the button updates the date and time displayed in the pagelet. (The refresh button in the header is an optional feature available in Oracle WebCenter Interaction only, configured on the Advanced Settings page of the Web Service editor.)



Current time is:
Wed May 11 2005 15:02:34 GMT-0700
(Pacific Standard Time)

The in-place refresh is executed by calling the `refresh()` method on the pagelet object instance. The pagelet reference can be retrieved by GUID, ID or name, available via the Oracle WebCenter Interaction Development Kit (IDK) `IPortletRequest` interface. You can also set a new URL to be displayed within the pagelet upon refresh by using `setRefreshURL` or passing in a new URL when you call `refresh`. (The title bar cannot be altered on refresh.)

```
<div style="padding:10px;" align="center">
<p><button onclick="refresh_portlet()">Refresh Portlet</button></p>
<p><b>Current time is:</b><br> <span id="refreshTimeSpan"></span></p>
</div>
<pt:namespace pt:token="$PORTLET_ID$"
xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'/>

<script type="text/javascript">
function refresh_portlet()
{
var refreshPortlet = PTPortlet.getPortletByID($PORTLET_ID$);
if (!refreshPortlet)
  {
  refresh_debug('Could not locate PTPortlet object which corresponds to <b>Refresh
Portlet</b> on page.');
  return;
  }
refresh_debug('<b>Refresh Portlet</b> calling refresh() method.');
refreshPortlet.refresh();
}

function refresh_debug(str)
{
if (window.PTDebugUtil)
  {
  PTDebugUtil.debug(str);
  }
}

var t = new Date();
document.getElementById('refreshTimeSpan').innerHTML = t;
</script>
```

### 2.2.4 Adaptive Pagelet Development Tips

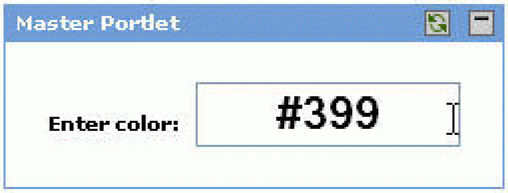
These tips apply to most adaptive pagelets.

- **Proxt all URLs.** You cannot make request to a URL whose host/port differs from that of the calling page. All URLs requested through JavaScript must be proxied. For details, see .Section 1.3, "About Server Communication and the Proxy".
- **Add all required JavaScript to the page in advance.** Browsers might not process script blocks/includes added to the page through the `innerHTML` property.
  - Microsoft Internet Explorer: Add the defer attribute to the script tag.
  - Netscape: Use RegExp to parse the response and look for the script, then eval it.
- **JavaScript HTTP and proxied HTTP must use the same authentication credentials.** JavaScript brokered HTTP requests send the same authentication token (cookie) as when you make a normal gatewayed HTTP request.

## 2.3 Session Preferences

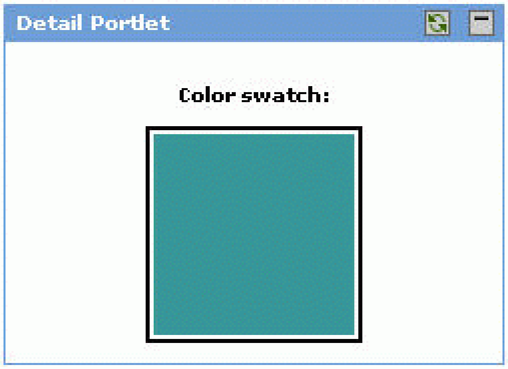
To store and share settings within the client browser, use session preferences.

Pagelets can use preferences to communicate with each other, but accessing preferences usually requires a round trip to a database. Session preferences provide a way to store and share settings in the user's session within the client browser. The Master-Detail design pattern illustrates the most basic usage of session preferences. This design pattern splits control and display between two pagelets. For example, the "master" pagelet could summarize data in list form, and the "detail" pagelet could display details on each data item in response to user selection. In the example below, the master pagelet displays a form that allows you to enter a color code in a text box.



Enter color: #399

When the user enters a color code in the text box, the color in the detail pagelet changes.



For each onkeyup event that occurs in the "Enter color" text box in the master pagelet, the following steps are executed:

1. The master pagelet sets the session preference using the current value of the text box.
2. The master pagelet calls an update method on the detail pagelet.
3. The detail pagelet retrieves the session preference to get the color value.
4. The detail pagelet redraws its color swatch area to reflect the new color value.

Pagelets can manipulate session preferences using the Oracle WebCenter Interaction Development Kit (IDK) or the Oracle WebCenter Interaction Scripting Framework. Sample code for both options is provided below.

---

**Note:** Shared session preferences must be specified by name on the Preferences page of the associated Web Service editor or they will not be sent to the pagelet.

---

## 2.3.1 Using Oracle WebCenter Interaction Development Kit Methods to Access Session Preferences

Always use the Oracle WebCenter Interaction Development Kit (IDK) to read session preferences, as shown in the example code below. (In most cases, reading session preferences via the Oracle WebCenter Interaction Scripting Framework is inefficient and insecure.)

**Java**

```
<%@ page language="java" import="com.plumtree.remote.portlet.*,java.util.Date" %>

IPortletContext portletContext =
PortletContextFactory.createPortletContext(request,response);
IPortletResponse portletResponse = portletContext.getResponse();
IPortletUser portletUser = portletContext.getUser();
IPortletRequest portletRequest = portletContext.getRequest();

masterColor = portletRequest.getSettingValue(SettingType.Session, "masterColor");
```

**.NET**

```
...
Dim portletContext As IPortletContext
portletContext = PortletContextFactory.CreatePortletContext(Request, Response)

Dim portletRequest As IPortletRequest
portletRequest = PortletContext.GetRequest

Dim portletUser As IPortletUser
portletUser = PortletContext.GetUser

Dim portletResponse As IPortletResponse
portletResponse = PortletContext.GetResponse

Dim masterColor As String
masterColor = portletRequest.GetSettingValue(SettingType.Session "masterColor")
...
```

### 2.3.2 Using Oracle WebCenter Interaction Scripting Framework Methods to Access Session Preferences

As noted above, it is usually better to  use the Oracle WebCenter Interaction Development Kit (IDK) to read session preferences; reading session preferences via the Oracle WebCenter Interaction Scripting Framework is inefficient and insecure.

This example is oversimplified; the master pagelet makes a direct call to a JavaScript method of the detail pagelet. Unless the master pagelet takes extra measures to ensure that the detail pagelet is actually present on the same page, calls from master to detail could generate errors. The Oracle WebCenter Interaction Scripting Framework provides an easy way to detach the relationship between pagelets and use a common event interface for communication.   For more information on the Oracle WebCenter Interaction Scripting Framework, see Section 2.2.3, "Oracle WebCenter Interaction Scripting Framework".

**Master Pagelet**

```
<div style="padding:10px;" align="center">
<p><b>Enter color:</b> &nbsp;
<input type="text" style="font-size:22px;font-weight:bold;text-align:center;"
id="master_prefName"
value="#FFFFFF" size="8" onkeyup="master_setPrefs(this.value)"></p><br>
</div>

<script type="text/javascript">
function master_setPrefs(val)
```

```
{
var prefName = 'masterColor';
var prefValue = val;
PTPortlet.setSessionPref(prefName,prefValue);

master_debug('<b>Master Portlet</b> called
PTPortlet.setSessionPref(\'masterColor\',\'' + prefValue + '\').');

if (window.detail_update)
  {
  master_debug('<b>Master Portlet</b> calling detail_update().');
  detail_update();
  }
else
  {
  master_debug('Could not locate portlet <b>Detail Portlet</b> on page.');
  }
}
function master_debug(str)
{
if (window.PTDebugUtil)
  {
  PTDebugUtil.debug(str);
  }
}
</script>
```

### Detail Pagelet

```
<div style="padding:10px;" align="center">
<p><b>Color swatch</b> &nbsp;
<div style="width:100px;height:100px;border:2px solid
black;padding:2px;"id="detail-swatch"></div>
<script>
function detail_update()
{
var color = PTPortlet.getSessionPref('masterColor');
detail_debug('<b>Detail Portlet</b> received value="' + color + '" for
PTPortlet.getSessionPref(\'masterColor\')');

var swatch = document.getElementById('detail-swatch');
if (swatch)
  {
  swatch.innerHTML = '<div style="background-color:' + color +
';width:100%;height:100%;"></div>';
  }
else
  {
  detail_debug('<b>Detail Portlet</b> cannot find \'detail-swatch\' DIV
element.');
  }
}

function detail_debug(str)
{
if (window.PTDebugUtil)
  {
  PTDebugUtil.debug(str);
  }
}
</script>
```

## 2.4 Pagelet Caching

Caching is the functionality that allows  Oracle WebCenter Ensemble to request pagelet content, save the content, and return the saved content to users when appropriate. The importance of caching cannot be overstated.

Efficient caching makes every web application faster and less expensive. The only time content should not be cached is if the data must be continuously updated. If every pagelet had to be freshly generated for each request, performance could become unacceptably slow. Oracle WebCenter Ensemble relies on caching to improve performance. pagelet content is cached and returned when later requests match the cache's existing settings.

Caching is indexed on the settings sent by the pagelet. When the Oracle WebCenter Ensemble proxy server processes a request for a page, it looks individually at each pagelet on the page and checks it against the cache. The process can be summarized as follows:

1. The proxy server assembles a cache key used to uniquely identify each pagelet in the cache.
2. The proxy server checks the cache for a matching cache key entry:
   - If the proxy erver finds a match that is not expired, it returns the content in the cache and does not make a request to the external resource.
   - If there is no matching cache key for the pagelet or if the cache key has expired, the proxy server makes a request to the external resource. If the matching cache entry uses ETag or Last-Modified caching, it also sends the appropriate caching header to the external resource in the request.
3. The response comes back from the external resource; the proxy server checks for caching headers:
   - If the headers include an Expires header, the proxy server stores the new pagelet content (along with a new expiration date) in its cache.
   - If the headers use ETag or Last-Modified caching, the existing cache entry might be revalidated (in the case of '304-Not Modified') or new pagelet content might be stored in the cache.

Oracle WebCenter Ensemble caches proxied content to complement, not replace, browser caching. Public content is accessible to multiple users without any user-specific information (based on HTTP headers). The proxy server calculates the cache headers sent to the browser to ensure that the content is properly cached on the client side.

Oracle WebCenter Ensemble caches all text (i.e., nonbinary) content returned by GET requests. Even if proxy caching is disabled (via PTSpy), pagelet caching still takes place. Proxied content can be cached by an external proxy server or by the user's browser. Beware browser caching of proxied content; it is a good idea to clear your browser cache often during development. An incorrectly set Expires header can cause browsers to cache proxied content.

The pagelet cache contains sections of finished markup and sections of markup that require further transformation. Post-cache processing means content can be more timely and personalized. Adaptive tags enable certain pagelets (for example, Community banners) to be cached publicly for extended periods of time and yet contain user specific and page-specific information, as well as the current date and time.

For details, see the following sections:

- Section 2.4.1, "About Pagelet Caching Strategies"
- Section 2.4.2, "Pagelet/Cache Key"
- Section 2.4.3, "Setting HTTP Caching Headers - Cache-Control"
- Section 2.4.4, "Setting HTTP Caching Headers - Expires"
- Section 2.4.5, "Setting HTTP Caching Headers - Last-Modified and ETag"

For a full explanation of HTTP caching, see RFC 2616 (`http://www.w3.org/Protocols/rfc2616/rfc2616.html`).

### 2.4.1 About Pagelet Caching Strategies

Pagelet caching is controlled both by the programmer and by the administrator who registers the pagelet in Oracle WebCenter Ensemble. Each and every pagelet needs a tailored caching strategy to fit its specific functionality.

A pagelet's caching strategy should take all possibilities into account and use the most efficient combination for its specific functionality. A pagelet that takes too long to generate can degrade the performance of every page that displays it. These questions can help you determine the appropriate caching strategy:

- Will the content accessed by the pagelet change? How often?
- How time-critical is the content?
- What processes are involved in producing pagelet content? How expensive are they in terms of server time and impact?
- Is the pagelet the only client with access to the back-end application?
- Is the content different for specific users?
- Can users share cached content?

Determine how often pagelet content must be updated, dependent on data update frequency and business needs. Find the longest time interval between data refreshes that will not negatively affect the validity of the content or the business goals of the pagelet.

Since caching is indexed on the settings used by a pagelet, new content is always requested when settings change (assuming that no cached content exists for that combination of settings).

There are two common situations in which you might mistakenly decide that a pagelet cannot be cached:

- **In-place refresh**: You might think that caching would "break" a pagelet that uses in-place refresh because the pagelet would be redirected to the original (cached) content. This can be avoided if a unique setting is updated on every action that causes a redraw, effectively "flushing" the cache. (In-place refresh renews the pagelet display by causing the browser to refresh page at a set interval.)
- **Invisible preferences**: If the content of the pagelet is dependent on something other than preferences (for example, the pagelet keys off the User ID to display a name or uses portal security to filter a list), caching can still be implemented with "invisible preferences" (in this case, User ID). As with in-place refresh, invisible preferences are set solely for the purpose of creating a different cache entry. They are set programmatically, without the user's knowledge.

For details on implementing caching, see the following sections:

- Section 2.4.2, "Pagelet/Cache Key"

- Section 2.4.3, "Setting HTTP Caching Headers - Cache-Control"
- Section 2.4.4, "Setting HTTP Caching Headers - Expires"
- Section 2.4.5, "Setting HTTP Caching Headers - Last-Modified and ETag"

## 2.4.2 Pagelet/Cache Key

The cache key for a pagelet entry in Oracle WebCenter Ensemble consists of the values in the table below.

| **Type** | **Value** |
|---|---|
| Pagelet ID | The unique ID for the pagelet, defined by Oracle WebCenter Ensemble. |
| Content Mode | The content mode of the pagelet. |
| Settings | Any settings stored in Oracle WebCenter Ensemble. |
| User Interface | The type of device used to access the pagelet. |
| LocaleID | The ID for the locale associated with the current user, defined by Oracle WebCenter Ensemble. |
| UserID | The unique ID for the current user. Included only if private caching is used. |
| URI | The URL to the current page on the external resource. |
| Last-modified date | The last modified date of the pagelet. |

The data below is deliberately not included in the cache key:

| **Type** | **Value** |
|---|---|
| StyleSheetURI | Stylesheets are applied at runtime, depending on the user preference. Pagelet content does not depend on the particular stylesheet that the user has selected. |
| HostpageURI | All parts of the Hostpage URI value are covered separately. The User ID is added if private caching is used. |

## 2.4.3 Setting HTTP Caching Headers - Cache-Control

The Cache-Control header can be used to expire content immediately or disable caching altogether. The value of this header determines whether cached pagelet content can be shared among different users.

The Cache-Control header can contain the following values:

| **Value** | **Description** |
|---|---|
| public | Allows any cached content to be shared across users with identical sets of preferences using the same Oracle WebCenter Ensemble server. This value should be used whenever possible. |
| private | Tells Oracle WebCenter Ensemble not to share cached content. The User ID is added to the cache key so that a separate copy is retained in the cache for each individual user. This value should only be used to protect sensitive information, for example, an e-mail inbox portlet. (User settings can also make public content effectively private.) |
| max-age=[seconds] | Specifies the maximum amount of time that an object is considered fresh. Similar to the Expires header, this directive allows more flexibility. [seconds] is the number of seconds from the time of the request that the object should remain fresh. |

| Value | Description |
|---|---|
| must-revalidate | Tells the cache that it must obey any freshness information it receives about an object. HTTP allows caches to take liberties with the freshness of objects; specifying this header tells the cache to strictly follow your rules. |
| no-cache | Disables caching completely and overrides Oracle WebCenter Ensemble settings. Neither the client nor Oracle WebCenter Ensemble responds to subsequent requests with a cached version. |

In JSP, use the `setHeader` method to configure the Cache-Control header:

```
<%
response.setHeader("Cache-Control","public");
%>
```

The JSP example below expires the content immediately using the maximum age header.

```
<%
response.setHeader("Cache-Control","max-age=0");
%>
```

In .NET, the Cache-Control header is accessed through the `System.Web.HttpCachePolicy` class. To set the header to public, private or no-cache, use the `Response.Cache.SetCacheability` method.

```
Response.Cache.SetCacheability(HttpCacheability.Public);
```

To set a maximum age for content in .NET, use the `Response.Cache.SetMaxAge` method. The example below expires the content immediately.

```
TimeSpan ts = new TimeSpan(0,0,0);
Response.Cache.SetMaxAge(ts);
```

To set the header to must-revalidate in .NET, use the `Response.Cache.SetRevalidation` method.

```
Response.Cache.SetRevalidation(HttpCacheRevalidation.AllCaches);
```

## 2.4.4 Setting HTTP Caching Headers - Expires

The Expires header specifies when content will expire, or how long content is "fresh." After this time, Oracle WebCenter Ensemble will always check back with the resource to see if the content has changed.

Most web servers allow setting an absolute time to expire, a time based on the last time that the client saw the object (last access time), or a time based on the last time the document changed on your server (last modification time).In JSP, setting caching to forever using the Expires header is as simple as using the code that follows:

```
<%
response.setDateHeader("Expires",Long.MAX_VALUE);
%>
```

The .NET `System.Web.HttpCachePolicy` class provides a range of methods to handle caching, but it can also be used to set HTTP headers explicitly (see MSDN for API documentation:
`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemwebhttpcachepolicyclasssetexpirestopic.as`

p). The `Response.Cache.SetExpires` method allows you to set the Expires header in a number of ways. The following code snippet sets it to forever:

```
Response.Cache.SetExpires(DateTime.Now.AddYears(100000000));
```

In .NET, the Web Form page (.aspx) can also use standard ASP methods to set HTTP headers.

> **Note:** **Never use Expires = 0 to prevent caching.** The Expires header is sent by the external resource and passed through to the browser by Oracle WebCenter Ensemble. Unless the time on all three machines is synchronized, an Expires=0 header can mistakenly return cached content. To solve this problem, set the Expires header to a fixed date that is definitely in the past.

### 2.4.5 Setting HTTP Caching Headers - Last-Modified and ETag

The Last-Modified response header specifies the last time a change was made in the returned content, in the form of a time stamp. ETag values are unique identifiers generated by the server and changed every time the object is modified. Either can be used to determine if cached content is up to date.

When an object stored in the cache includes a Last-Modified or ETag header, Oracle WebCenter Ensemble can use this value to ask the resource if the object has changed since the last time it was seen.

- Oracle WebCenter Ensemble sends the value from the Last-Modified header to the resource in the If-Modified-Since Request header.
- The resource sends the ETag header to Oracle WebCenter Ensemble with pagelet content. When another request is made for the same content, Oracle WebCenter Ensemble sends the value in the ETag header back to the resource in the If-None-Match header.

The pagelet code on the resource uses the header value to determine if the content being requested has changed since the last request, and responds with either fresh content or a 304 Not Modified Response. If Oracle WebCenter Ensemble receives the latter, it displays the cached content.JSP pagelets can access the value in the If-Modified-Since request header using the `getLastModified(HttpServletRequest req)` method provided by the Java class HttpServlet.In .NET, the `Response.Cache.SetLastModified` method allows you to set the Last-Modified header to the date of your choice. Alternately, the `SetLastModifiedFromFileDependencies` method sets the header based on the time stamps of the handler’s file dependencies.

```
Response.Cache.SetLastModified(DateTime.Now);
```

To use ETag in .NET, use the `Response.Cache.SetETag` method to pass in the string to be used as the ETag. The `SetETagFromFileDependencies` method creates an ETag by combining the file names and last modified timestamps for all files on which the handler is dependent.

## 2.5 Pagelet Internationalization

These tips and best practices apply to all pagelets that will be translated into multiple languages.

- **Identify ALL culturally dependent data.** Text messages are the most obvious example of locale-specific data, but there are many other parts of a service that can vary with language or location. These include: images, UI labels and buttons, icons, sounds, graphics, dates, times, measurements, honorifics and titles, phone numbers, and postal addresses.
- **Do not use compound messages (concatenated strings) to create text.** Compound messages contain variable data. For example, in the text string "You have XX credits," only the integer "XX" will vary. However, the position of the integer in the sentence is not the same in all languages. If the message is coded as a concatenated string, it cannot be translated without rewriting the code.
- **Use the IDK to avoid encoding issues.** All content is stored in the database in Unicode. The Oracle WebCenter Interaction Development Kit (IDK) handles encoding for international characters.

For details on implementing internationalization, see Section 2.2.2.2, "Using Internationalized Strings in Adaptive Tags".

## 2.6 Pagelet Configuration in Oracle WebCenter Ensemble

To deploy a pagelet in Oracle WebCenter Ensemble, you must configure Resource and Pagelet objects.

To deploy a pagelet, you must create and configure the following objects:

- Resource (producer): For details, see Section 2.6.1, "Configuring an Oracle WebCenter Ensemble Resource".
- Pagelet: For details, see Section 2.6.2, "Configuring an Oracle WebCenter Ensemble Pagelet".
- Resource (consumer): For details, see Section 2.6.1, "Configuring an Oracle WebCenter Ensemble Resource".

You must also configure security settings for Oracle WebCenter Ensemble objects. For details, see Section 2.6.5, "About Oracle WebCenter Ensemble Security".

To insert pagelets into a page, use one of the following methods:

- To insert a pagelet into a proxied page, see Section 2.2.2.4.1, "Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags".
- To insert a pagelet into a non-proxied page, see Section 2.6.4, "Inserting Pagelets into Non-Proxied Pages".

### 2.6.1 Configuring an Oracle WebCenter Ensemble Resource

Producer and consumer resources are both configured through the same editor.

- The **producer** resource defines the location of the web application that implements the pagelet code and how it is accessed. In most cases, a single producer resource is used for multiple pagelets.
- The **consumer** resource defines the location of the web application that displays the pagelet code and how it is accessed.

To create a new resource, go to the **Applications** section of the **Ensemble Console**, click the **Resources** sub-tab, and click **Create New**.

To edit an existing resource, navigate to the **Resources** sub-tab and click the resource name.

The resource configuration editor includes the tabs detailed below.

### General

The following settings are configured on the General tab:

- **Resource Attributes: Name**: The resource name displayed in other sections of the Oracle WebCenter Ensemble Console. This name must be unique. The **Description** is optional.
- **Resource Attributes: Owner**: Set to the user who created the resource and can only be changed by an administrator.
- **Timeout**: This setting applies to the resource and any associated pagelets. By default, the timeout is set to 30 seconds, but can be extended if necessary.
- **Status**: This option allows you to disable a resource if it is not accessible.
- **Is Login resource**: This option defines the resource as a login resource, used solely to authenticate users. Login resources are not protected by policies, because they must be accessible by all users at all times. For details on using login resources, see Chapter 3, "Oracle WebCenter Ensemble Login Customization".
- **Policy Attributes: Name**: The name of the policy set associated with the resource. This name defaults to the name of the resource, but can be modified.
- **Policy Attributes: Owner**: Set to the user who created the resource and can only be changed by an administrator.

### Connections

The Connections tab allows you to define the **Internal URL prefix** that the Oracle WebCenter Ensemble proxy uses to access the application, and an **External URL prefix** to be exposed to users. The external URL prefix may be absolute or relative to the Oracle WebCenter Ensemble proxy. If the URL is absolute, it must either point directly to the Oracle WebCenter Ensemble proxy or, be resolvable to the Oracle WebCenter Ensemble proxy through DNS. You may configure multiple external URL prefixes to map to the internal URL prefix.

The **Enable URL Rewriting** option allows you to choose whether or not links within a proxied application will be transformed. If the internal and external URL prefixes are identical or all application links are relative, disabling URL rewriting will improve performance.

### Credential Mapping

The Credential Mapping tab allows you to configure a resource to log in a user to the back-end application automatically by supplying the required authentication credentials. To disable **Resource Authentication**, select Disabled.

This page can be used to configure **HTML Form-based authentication** or **Basic authentication** using static login credentials, user profile information, or the credential vault. For detailed instructions, see the online help and the*Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* "Chapter 6, "Credential Mapping."

### Headers

This tab allows you to choose which **request headers** and **response headers** are passed on to the back-end application. For example, if you are using delegated authentication, the SSO system might insert headers that should not be passed to the back-end application.

### CSP

CSP is the protocol used by Oracle WebCenter Interaction to communicate with external resources. The CSP tab allows you to specify whether or not a **login token** will be sent to the back-end application. The login token is necessary to use the Oracle WebCenter Interaction (IDK) Remote APIs. For details, see the *Oracle Fusion Middleware Web Service Developer's Guide for Oracle WebCenter Interaction*.

This page also allows you to specify **session preferences** that can be set or obtained from the application and which **user information** will be sent to the application. For details on session preferences, see Section 2.3, "Session Preferences"

**Roles**

This tab allows you to specify the **Roles** used by the application. The policy set associated with the resource maps users and groups to these roles via policies and policy rules. For details, see Section 2.6.5.1, "Using Oracle WebCenter Ensemble Roles in Pagelets and Proxied Applications".

**SSO Log Out Settings**

This tab allows you to enter an URL pattern to trigger Oracle WebCenter Ensemble SSO log out functionality. A user can be logged in to multiple applications through Oracle WebCenter Ensemble via a single sign on (SSO) system. When a user logs out of an application, Oracle WebCenter Ensemble can prompt the user to log out of that application or all applications. To enable SSO log out for a resource, enter the pattern of the logout URL in the **Internal log out URL patterns** list. You may enter multiple patterns.

For more information on resource configuration, see the online help and the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* "Chapter 4: Proxy Resources".

## 2.6.2 Configuring an Oracle WebCenter Ensemble Pagelet

The pagelet object defines the location of the pagelet file and any associated parameters.

To create a new pagelet, go to the **Applications** section of the Oracle WebCenter Ensemble Console, click the **Pagelets** sub-tab, and click **Create New**.

To edit an existing pagelet, navigate to the **Pagelets** sub-tab and click the pagelet name. Pagelets can be created and edited by administrators, managers, and resource owners. Resource owners can only create or edit pagelets if they own the associated resource.

The pagelet configuration editor includes the tabs detailed below.

**General**

The following settings are configured on the General tab:

- **Name**: The pagelet name displayed in other sections of the Oracle WebCenter Ensemble Console. This name must be unique. The **Description** is optional.
- **Parent resource**: The producer resource associated with the pagelet. Each pagelet must be associated with a resource. Multiple pagelets can be associated with the same resource.
- **Library**: A user-defined way to group pagelets. To add the pagelet to a pagelet library, type the name of the pagelet library. If the pagelet library does not already exist, one will be created.
- **Sample code**: This field displays the XML declaration for the pagelet (pt.ensemble.inject) that can be used to insert the pagelet in a consumer resource page. This field is empty until the pagelet is saved. For information on using this

code in a pagelet, see Section 2.2.2.4.1, "Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags".

- **Publish documentation**: This option allows you to choose whether the pagelet is included in the Developer Pagelet Catalog.
- **Add inline refresh to all URLs**: This option allows you to enable automatic inline refresh for the pagelet and related URLs. If you select this option, make sure to specify an appropriate **Refresh interval**.

**Location**

The pagelet location is composed of the **Internal URL prefix** of the associated resource and an URL suffix that points to the pagelet application hosted by that resource. The internal URL prefix is defined by the associated resource; to modify it, edit the resource.

**Parameters**

The Parameters tab provide access to configuration settings for data transport.

The **Payload schema URL** allows you to apply an XML schema to the pagelet's payload. Oracle WebCenter Ensemble only supplies the URL to the pagelet; it is up to the pagelet to use the schema to validate the XML payload. For information on using payloads, see the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* "Chapter 9, "Pagelets".

**Parameters** are name/value pairs (attributes) that provide information to the pagelet. This page allows to set a name and type for each parameter and select whether the parameter is mandatory.

The **Pagelet Parameter Transport Type** allows you to port Oracle WebCenter Interaction portlets that use Administrator, CommunityPortlet, or Community level preference settings to work as pagelets within Oracle WebCenter Ensemble. To send attributes to a portlet as an Oracle WebCenter Interaction setting type, choose the transport type associated with the setting type and enter the preference names in the parameters list. By default, attributes are send in the HTTP request header. Pagelet parameter values are defined in the pagelet injection code that is added to a consumer page. For information on using parameters in a pagelet, see the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* "Chapter 9, "Pagelets".

**Metadata**

Metadata can be used to store additional information about a pagelet. **Metadata** fields are viewable in the pagelet documentation.

**Consumers**

By default, all resources are allowed to consume a pagelet. This tab allows you to restrict which resources are allowed to consume the pagelet. To limit access to the pagelet, clear the **All consumers allowed** check box and add any resources that should have access to the pagelet to the **Consumers** list.

For more information on pagelet configuration, see the Oracle WebCenter Ensemble online help and the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* "Chapter 9, "Pagelets". For details on configuring pagelets and portlets in Oracle WebCenter Interaction, see the Oracle WebCenter Interaction online help and the *Oracle Fusion Middleware Web Service Developer's Guide for Oracle WebCenter Interaction*

### 2.6.3 Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags

The `pt:ensemble.inject` tag injects the output of the specified pagelet into the page.

The pagelet is referenced by the fully qualified name of the pagelet as defined in Oracle WebCenter Ensemble, in the form `libraryname:pageletname`. Any attributes not prefixed with "pt:" will be passed on to the pagelet. Any HTML content inside the pagelet tag will be passed to the pagelet as an XML payload.

```
<pt:ensemble.inject pt:name="mylibrary:mypagelet" pagelet-attribute="A pagelet
attribute">
<?xml version="1.0" encoding="utf-8"?>
<doc>This is an XML payload.</doc>
</pt:ensemble.inject>
```

For an example of using this code, seeSection 2.1.1, "Creating a Custom Pagelet with the Java Oracle WebCenter Interaction Development Kit (IDK) Proxy API" or Section 2.1.2, "Creating a Custom Pagelet with the .NET Oracle WebCenter Interaction Development Kit (IDK) Proxy API".You can also insert pagelets into non-proxied pages; for details, see Section 2.6.4, "Inserting Pagelets into Non-Proxied Pages".

### 2.6.4 Inserting Pagelets into Non-Proxied Pages

You can also insert pagelets into non-proxied pages using a simple javascript function.

To activate this feature, add the following HTML snippet in the <HEAD> section of the page.

```
<script type="text/javascript" src="http://proxy:port/inject/v2/csapi">
</script>
```

This script injects all CSAPI and pagelet inject functions into the page to display the pagelet. One of the sections injected is the following function:

```
function injectpagelet(library, name, injectmethod, payload, arguments)
{
    ...
}
```

This function injects an Oracle WebCenter Ensemble pagelet as a widget into the parent page. The method interface is as follows:

- **library**: The library name of the pagelet to inject. This argument is a string that accepts spaces like 'library name'.
- **name**: The name of the pagelet to inject. This argument is a string that accepts spaces like 'pagelet name'.
- **injectmethod**: Specifies the manner of injecting the pagelet, if set to 'iframe', then an iframe is inserted where the call occurs, otherwise inline HTML will be used. If the inject method is specified as 'iframe', then a set of IFrame options can be sent along. These options control how the iframe will be displayed. The following iframe options are supported: width, height, frameborder, align, longdesc, marginheight, marginwidth, scrolling, stylem class. The parameters are given in the form of `param=value`. The separator between parameters are spaces. For example, `'iframe align=right frameborder=1 width=100% height=200 scrolling=yes class=myclass'`.
- **payload**: The XML payload to send along with the pagelet request
- **arguments**: The pagelet arguments to send along with the pagelet request. Should be given in the form of: `'param1=value1&param2=value2&param3=value3'`.

The script also creates a new <div> with a unique name that includes a reference to the `injectpagelet` function. Several examples are shown below:

```
<div>
    <script type="text/javascript">
        injectpagelet('library', 'name');
    </script>
</div>
<div>
    <script type="text/javascript">
    injectpagelet('library', 'name', 'iframe', 'payload',
'param1=value1&param2=value2&param3=value3');
    </script>
</div>

<div>
    <script type="text/javascript">
    injectpagelet('library', 'name', 'iframe width=100% height=200', 'payload');
    </script>
</div>
```

### 2.6.4.1 Using Automatic Resizing with IFrames

The pagelet inject function can automatically resize the IFrame that encapsulates pagelet content. The resizing is done so that the IFrame stretches to fit the content within. To use this feature, the ifwidth and ifheight parameters must be set to 'auto' as shown in the example below:

```
<script type="text/javascript">
injectpagelet('library', 'pagelet', 'iframe ifheight=auto ifwidth=auto');
</script>
```

In addition, this feature relies on an external page on the same domain as the consumer page. This page is included into the pagelet IFrame as an internal hidden IFrame. This page collects the sizing information and passes it on to the parent consumer page. This page must be deployed in the same directory as the consumer page. An example is shown below.

```
<html>
  <head>
    <title>Resizing Page</title>
    <script type="text/javascript">
function onLoad() {
var params = window.location.search.substring( 1 ).split( '&' );
var height;
var width;
var iframe;

for( var i = 0, l = params.length; i < l; ++i ) {
var parts = params[i].split( '=' );
switch( parts[0] ) {
case 'height':
height = parseInt( parts[1] );
break;
case 'width':
width = parseInt( parts[1] );
break;
case 'iframe':
iframe = parts[1];
break;
}
}
window.top.updateIFrame( iframe, height, width );
}

if (window.addEventListener) {
window.addEventListener("load", onLoad, false)
} else if (window.attachEvent) {
window.detachEvent("onload", onLoad)
window.attachEvent("onload", onLoad)
} else {
window.onload=onLoad
}
    </script>
  </head>
  <body>
  </body>
</html>
```

To insert a pagelet into a proxied page, use the `pt:inject` tag; for details, see Section 2.2.2.4.1, "Inserting Pagelets Using Oracle WebCenter Ensemble Adaptive Tags". You can also inject pagelets using REST; for details, see Chapter 4, "Oracle WebCenter Ensemble REST APIs".

## 2.6.5 About Oracle WebCenter Ensemble Security

There are two factors that control access to a resource in Oracle WebCenter Ensemble: authentication and policies.

Oracle WebCenter Ensemble manages authentication with each proxied application based on the settings defined for the associated resource.

You can define the credential mappings for each resource. For details on configuring credential mappings, see the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* and the online help. You can create custom mappings to external credential stores. For details, see Section 2.6.5.2, "Creating a Custom Credential Mapping".

Each resource is protected by a policy set, which describes the conditions under which a user may be granted access to the resource, and the roles associated with those conditions. For details on using roles, see Section 2.6.5.1, "Using Oracle WebCenter Ensemble Roles in Pagelets and Proxied Applications".

### 2.6.5.1 Using Oracle WebCenter Ensemble Roles in Pagelets and Proxied Applications

Pagelets and proxied applications can use Oracle WebCenter Ensemble roles to control access to content and functionality.

Each incoming request to Oracle WebCenter Ensemble is evaluated against the policies for the requested resource. If the user is found to be in one or more roles, access is granted and the set of matching roles is passed on to the proxied application, allowing the application to determine the correct access level for the user. This is called **Role-Based Access Control (RBAC)**.Roles are sent in an HTTP header and can be accessed using the Proxy IDK and adaptive tags.**Adaptive tags** can be included in the markup returned by any proxied page, including pagelets. Using the attributes defined in the tag, Oracle WebCenter Ensemble transforms the XML and replaces it with standard HTML to be displayed in a browser. For details, see Section 2.2.2.4, "Oracle WebCenter Ensemble Adaptive Tag Library (pt:ensemble)".

- The `pt:ensemble.rolelist` tag creates a collection of the user's roles in the current context and stores it in memory using the name in the pt:key attribute.

Each item in the collection is a variable containing the role name. The example below displays a list of the user's roles by iterating over the collection using the pt:logic.foreach tag.

```
<pt:ensemble.rolelist pt:key='roles'/>
<pt:logic.foreach pt:data='roles' pt:var='role'>
<pt:logic.value pt:value='$role'/>
 <pt:logic.separator><br></pt:logic.separator>
</pt:logic.foreach><BR>
```

- The `pt:ensemble.roleexpr` tag evaluates an expression and stores the result as a boolean in memory using the name in the pt:key attribute. The example below checks if the user has the Admin role and displays a message based on the result using the pt:logic.if tag.

```
<pt:ensemble.roleexpr pt:expr='hasRole Admin' pt:key='hasrole'/>
<pt:logic.if pt:expr='$hasrole'>
   <pt:logic.iftrue>
      This user has the Admin role.
   </pt:logic.iftrue>
   <pt:logic.iffalse>
      Warning: This user DOES NOT have the Admin role.
   </pt:logic.iffalse>
</pt:logic.if>
```

The IDK `bea.alui.proxy.IProxyUser` interface also allows you to get a list of the user's roles in the current context, or determine whether the user has a specific role.

- The `IProxyUser.getRoles` method returns an iterator of the user's roles as strings.
- The `IProxyUser.isUserInRole` method determines whether the user is in the role passed in the role parameter and returns true if the user has the role (false otherwise).
- The `IProxyUser.isAnonymous` method determines whether the user is an Anonymous user.
- The `IProxyUser.isUserInRole` method determines whether the user is in the role passed in the role parameter and returns true if the user has the role (false otherwise).

The simplified example below (roleconsumer.jsp) retrieves role information for the current user. The associated Oracle WebCenter Ensemble resource has three roles defined: AdminRole, MgrRole, and UserRole. (The associated policy set assigns these roles to groups or users.) In this example, the associated Oracle WebCenter Ensemble pagelet is named 'rolePagelet'. For more details on the Oracle WeCenter Interaction Development Kit (IDK) proxy API, see the API documentation.

```
<%@ page language='java' import='com.plumtree.remote.portlet.*, java.util.Date,
java.util.*, com.bea.alui.proxy.*' %>

You refreshed at <%= new Date().toString()%><br/>
<%
response.setHeader('Cache-Control','no-cache');  //HTTP 1.1
response.setHeader('Pragma','no-cache');  //HTTP 1.0
response.setDateHeader ('Expires', 0);  //prevents caching at the proxy server

IProxyContext ctx =
ProxyContextFactory.getInstance().createProxyContext(request,response);
IProxyRequest req = ctx.getProxyRequest();
IProxyResponse res = ctx.getProxyResponse();
```

```
Enumeration roles = req.getUser().getRoles();
boolean isAdmin = req.getUser().isUserInRole('AdminRole');
boolean isMgr = req.getUser().isUserInRole('MgrRole');
boolean isUser = req.getUser().isUserInRole('UserRole')
%>

<html>
<head>
<meta http-equiv='Content-Type' content='text/html; charset=ISO-8859-1'>
<META HTTP-EQUIV='PRAGMA' CONTENT='NO-CACHE'>
<title>Preferences</title>
</head>

<body>
<br/> CONSUMER SETTINGS <br/>
<% while (roles.hasMoreElements()) {
   String role = (String)roles.nextElement();  %>
   <br/>User has role: <%=role%><br/>
<% } %>
<br/>User is admin? <%=isAdmin%><br/>
<br/>User is manager? <%=isMgr%><br/>
<br/>User is standard user? <%=isUser%><br/>

<pt:ensemble.inject xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'
pt:name='idkLib:rolePagelet'/>

</body>
</html>
```

#### 2.6.5.2 Creating a Custom Credential Mapping

Oracle WebCenter Ensemble provides an API for creating custom mappings to external credential stores, allowing you to authenticate users against a custom credential source.

The `IVendorCredentialMapper` interface defines the Oracle WebCenter Ensemble interface for objects capable of obtaining an appropriate set of credentials needed for secondary authentication for a particular user in an application. To implement this interface, follow the directions below.

1. Create a java class that implements the`com.plumtree.runner.credentialmapper.IVendorCredentialMapper` interface.
2. Map the `getCredential` and `setCredential` methods of this interface to your credential vault. The simplified example below uses an internal class called `VConnector` and `calls VConnector.getInstance().getCrededentialsForDomain`. Note: This step is vendor-specific. It will probably include a network hop, since the credential store will most likely reside on another server. You must give the mapper a unique name, and localized ones if necessary. See the IVendorCredentialMapper API documentation for all required names.
3. Compile the class into a jar. The build process must link to common.jar, included with the Oracle WebCenter Ensemble distribution.
4. To load the custom vault into Oracle WebCenter Ensemble, copy the jar file to the Oracle WebCenter Ensemble server and edit the configuration.xml file. Add the following component, and include the path to the custom jar file in the `<value>` element:

```
<component name="runner:credentialproviders"
type="http://www.plumtree.com/config/component/type/credentialproviders">
    <setting name="CredentialVaultClassPath">
        <value xsi:type="xsd:string">c:/jarfolder/jarname.jar</value>
    </setting>
    <clients>
        <client name="runnercontext" />
    </clients>
</component>
```

If the Oracle WebCenter Ensemble proxy and adminui run on different servers, the jar file must be copied to both servers, and the configuration.xml file on both servers must be edited.

5. Restart the Oracle WebCenter Ensemble server (both proxy and adminui if they are on separate servers). The custom credential vault should show up in the list of credential sources on the Credential Mapping page of the Resource editor.

The example below is simplified for illustration purposes.

```
package com.oracle.credentialvault;

import com.oracle.connector.CredentialsSet;
import com.oracle.connector.VConnector;
import com.plumtree.runner.credentialmapper.Credential;
import com.plumtree.runner.credentialmapper.IVendorCredentialMapper;

public class OracleCredentialVault implements IVendorCredentialMapper {

    /*
     * Ensemble will pass credential types as following:
     * Runner_*, where * is what the credential value type associated with this
login form in the Ensemble adminui.
     * For example, if the credential value type is 'username' then "Runner_
username" will be passed to the mapper.
     */
    public Credential getCredential(String initiator, String credType) {
        System.out.println("OracleCredentialVault::getCredential, initiator: " +
initiator + ", credType: " + credType);

        /*
         * Since this vault stores credentials per user and domain, we need to
devise a scheme to
         * map Ensemble's credential type to a domain. One way to do this is to
specify the credential
         * type as something like: "domain_type", which would translate to
credTypes like:
         * Runner_domain.com_username and Runner_domain.com_password
         */

        String username = initiator.toLowerCase(); // lets assume that the vault
stores all usernames in lowercase
        String domain = "oracle.com"; //getDomain(credType); // lets assume that
the vault stores all domains in lowercase
        String type = credType; //getType(credType);

        CredentialsSet credSet =
VConnector.getInstance().getCrededentialsForDomain(username, domain);
        if( credSet != null ) {
            System.out.println("OracleCredentialVault::getCredential, found vault
set: " + credSet.toString() + ", returning type = " + type);
```

```
            return new Credential(credSet.getCredential(type));
        } else {
            System.out.println("OracleCredentialVault::getCredential, found null
vault set");
            return null;
        }

    }

    public String getDescription(String userLocale) {
        return "Test mapper that mimics a mapper between Ensemble and a credential
vault that associates credentials with a username/domain relationship";
    }

    public String getName() {
        return "OracleCredentialVault";
    }

    public String getName(String userLocale) {
        return "OracleCredentialVault";
    }

    public String getVendorName(String userLocale) {
        return "Oracle";
    }


    public boolean setCredential(String initiator, Credential credential, String
credType) {
        System.out.println("OracleCredentialVault::setCredential, initiator: " +
initiator + ", credType: " + credType + ", Credential: " +
credential.getCredentialValue());

        String username = initiator.toLowerCase(); // lets assume that the vault
stores all usernames in lowercase
        String domain = "oracle.com"; //getDomain(credType); // lets assume that
the vault stores all domains in lowercase
        String type = credType; //getType(credType);

        System.out.println("OracleCredentialVault::setCredential setting username:
" + credential.getCredentialValue());
        CredentialsSet userCredSet =
VConnector.getInstance().getCrededentialsForDomain(username, domain);
        userCredSet.setCredential(type, credential.getCredentialValue());
        VConnector.getInstance().setCrededentialsForDomain(username, domain,
userCredSet);
        return true;

    }

    public boolean supportsCredentialsEditing() {
        // We can set new credentials using this vault
        return true;
    }

    /*
    private String getDomain(String credType) {
        int dstart = credType.indexOf("_");
        int dend = credType.indexOf("_", dstart+1);
        String domain = credType.substring(dstart+1, dend);
        System.out.println("TestMapper::getDomain, reading domain as: " + domain);
        return domain;
    }
    */

    /*
    private String getType(String credType) {
        int dstart = credType.indexOf("_");
        dstart = credType.indexOf("_", dstart+1);
        String type = credType.substring(dstart+1, credType.length());
        System.out.println("TestMapper::getType, reading type as: " + type);
        return type;
    }
    */

    /*
    private String doGetPropertyValue(String principal, String property) {
        return doGetPropertyValue(principal, property, ",", "=");
    }
    */

    /*
    private String doGetPropertyValue(String principal, String property, String
propDelim, String valueDelim) {
        int propertyindex =
principal.toLowerCase().indexOf(property.toLowerCase());
        String uname = null;
        if( propertyindex != -1) {
            // found a property occurence
            int beginIndex = propertyindex;
            int endIndex =
principal.toLowerCase().indexOf(propDelim.toLowerCase(), beginIndex);

            String prop = null;
            if(endIndex != -1) {
                prop = principal.subSequence(beginIndex,
endIndex).toString().trim();
            } else {
                prop = principal.subSequence(beginIndex,
principal.length()).toString().trim();
            }

            if( prop != null ) {
                int valueIndex = prop.toLowerCase().indexOf(valueDelim);
                if(valueIndex != -1) {
                    uname = prop.subSequence(valueIndex + valueDelim.length(),
prop.length()).toString().trim();
                }
            }

        }
        return uname;
    }
    */

}
```

For details on configuring resources to use credential mappings, see the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* and the online help.

# 3

# Oracle WebCenter Ensemble Login Customization

To display custom pages to the user at different steps during Oracle WebCenter Ensemble login and logout, you can customize a range of steps in the login process.

You can create custom login, logout, error and interstitial pages and configure Oracle WebCenter Ensemble to display them at specific points in the login/logout process. (An interstitial page is a page that appears before the expected content page.) Custom pages are hosted on a proxied application server, called the **login resource**. For details, see the following sections:

- Section 3.1, "Creating a Custom Oracle WebCenter Ensemble Pre-Login Page"
- Section 3.2, "Creating a Custom Oracle WebCenter Ensemble Login Page"
- Section 3.3, "Creating a Custom Oracle WebCenter Ensemble Error Page"
- Section 3.4, "Creating a Custom Oracle WebCenter Ensemble Post-Login Page"
- Section 3.5, "Creating a Custom Oracle WebCenter Ensemble Post-Logout Page"
- Section 3.6, "Configuring Custom Oracle WebCenter Ensemble Login Pages"
- Section 3.7, "Oracle WebCenter Ensemble Login Headers"

## 3.1 Creating a Custom Oracle WebCenter Ensemble Pre-Login Page

The pre-login page is an interstitial page displayed before the login form.

The pre-login page could display an important message about availability or new functionality. The pre-login page can also be used to display a custom message to users who are part of an experience definition that is blocked from accessing the requested resource. In the example below, the pre-login page (preinterstitialpage.jsp) displays a message about server maintenance. This page uses the `pt:common.error` tag to display any errors within the page, and the `pt:core.html` tag to display the submit button.

```
<%@page contentType="text/html;charset=UTF-8"%>
<HTML>
<BODY>
<SPAN xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<FORM action="./processpreinterstitialpage.jsp" method="POST">
<P>
<TABLE>
<TR><TD>
<CENTER><B>Maintenance Updates</B></CENTER>
</TD></TR>
```

```
<TR><TD>
 This server will be going down for maintenance on 5/28/07 at 6:37PM and 24
seconds.<BR>
<pt:common.error>
<P><B><FONT color="red"><pt:logic.value pt:value="$#10.ptmsgs_login"/></FONT>:</B>
<pt:common.errortext/>
</P>
</pt:common.error>
</TD></TR>
<TR><TD>
<CENTER>
<pt:core.html pt:tag="input" type="submit" value="$#2.ptmsgs_samples"/>
</CENTER>
</TD></TR>
</TABLE>
</FORM>
</SPAN>
</BODY>
</HTML>
```

When the user clicks the submit button, the processing page shown below (processpreinterstitialpage.jsp) sets the `runner_pre_interstitial_complete` header to **true**, which directs the browser to the login page. It also provides error text for the error page in case processing fails.

```
<%@page contentType="text/html;charset=UTF-8"%>
<HTML>
<BODY>
<%
out.println( "<P>Ensemble pre-login interstitial page processing login completion
error.
Contact your system administrator.</P>");
response.addHeader( "runner_pre_interstitial_complete", "true")
%>
</BODY>
</HTML>
```

## 3.2 Creating a Custom Oracle WebCenter Ensemble Login Page

The login page allows you to customize the Oracle WebCenter Ensemble login form display and functionality.

The `pt:ensemble.authsourcedata` tag provides a collection of the authentication sources available for the resource. The data is stored as a collection, and each item in the collection is a data object containing information about the authentication source (prefix, name, description) accessible through the data object dot notation (`$authsource.name`). You can use additional adaptive tags to iterate through the collection and allow the user to select the appropriate choice, as shown in the example that follows.

The example below (loginpage.jsp) displays a banner and a login form. The login form posts back to the page, which sets the appropriate headers to authenticate with the resource (`runner_username`, `runner_password`, `runner_authentication_provider`, and `runner_portal_authentication_source`). This page uses the `pt:ensemble.authsourcedata` tag, as well as several other adaptive tags to handle logic and display. For details on adaptive tags, see Section 2.2.2, "Adaptive Tags".

```
<%@ page import="java.net.*"%>
 <%@page contentType="text/html;charset=UTF-8"%>
<%
```

```
String username = request.getParameter( "username" );
String password = request.getParameter( "password" );
if(username != null && password != null)
{
   response.addHeader( "runner_username", username);
   response.addHeader( "runner_password", password);
}
String authsource = request.getParameter( "authsource" );
if ( authsource != null
{
   response.addHeader( "runner_authentication_provider", "portal");
   response.addHeader( "runner_portal_authentication_source", authsource );
}
%>
<html>
<head>
<link rel='stylesheet' type='text/css' href='css/main.css'/>
</head>
<body>
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<table class="banner" cellpadding="0" cellspacing="0">
<tr>
   <td class="appLogo"></htm:td>
   <td class="liquid"></htm:td>
 </tr>
</table>
<br/>

<!-- Welcome message: -->
<font face="arial" size=2> <pt:logic.value pt:value="$#3.ptmsgs_login"/><br>
<pt:logic.value pt:value="$#7.ptmsgs_login"/> </font>
<br/>
<%
String errorValue = request.getHeader("runner_error_last_error_message" );
if(errorValue != null)
{
   out.println("<br>");
   out.println("<font face=\"arial\" color=\"0000AA\" size=2>");
   out.println("<pt:logic.value pt:value=\"$#8.ptmsgs_login\"/><br>");
   out.println("</font>");
   out.println("<font face=\"arial\" color=\"AA0000\" size=2>");
   out.println("<b>");
   out.println(errorValue);
   out.println("</b>");
   out.println("</font>");
   out.println("<br>");
}
%>
<br/>
<pt:common.error>
<P><B><FONT color="red"><pt:logic.value pt:value="$#10.ptmsgs_login"/></FONT>:</B>
<pt:common.errortext/>
 </pt:common.error>

<!-- Login form: -->
<FORM name="loginform" action="loginpage.jsp" method="POST">
<table>
<tr><!-- Username -->
    <td align="right"><font face="arial" size=2>
    <pt:logic.value pt:value="$#0.ptmsgs_login"/>
    </font></td>
    <td align="left"><font face="arial" size=2>
    <!-- inputs and password inputs are of slightly different length in IE.  This
can be fixed with CSS.-->
    <pt:core.html pt:tag="input" type="text" name="username" alt="$#0.ptmsgs_
login" size="30" value="${param["username"]}"/>
    </font> </td> </tr>
<tr><!-- Password -->
    <td align="right"><font face="arial" size=2>
    <pt:logic.value pt:value="$#1.ptmsgs_login"/>
    </font></td>
    <td align="left"><font face="arial" size=2>
    <pt:core.html pt:tag="input" onkeypress="return submitform(event)"
type="password"
name="password" size="30" alt="$#1.ptmsgs_login" value="${param["password"]}"/>
    </font> </td> </tr>
 <!-- Auth sources -->
<pt:ensemble.authsourcedata pt:key="authsources"/>
<pt:logic.collectionlength pt:data="authsources" pt:key="authsourceslength"/>
<pt:logic.intexpr pt:expr="($authsourceslength)>0" pt:key="hasvalues"/>
<pt:logic.if pt:expr="$hasvalues">
<pt:logic.iftrue>
   <pt:logic.intexpr pt:expr="($authsourceslength)>1" pt:key="hasmultvalues"/>
   <pt:logic.if pt:expr="$hasmultvalues">
   <pt:logic.iftrue>
       <tr><!--Authentication Source:-->
          <td align="right" width="40%" colspan="1"><font face="arial" size=2>
          <pt:logic.value pt:value="$#5.ptmsgs_login"/>
          </font></td>
          <td align="left" width="40%" colspan="1"><font face="arial" size=2>
          <select name="authsource" onkeypress="return submitform(event)"
lang="en">
             <pt:logic.foreach pt:data="authsources" pt:var="auth">
                <pt:core.html pt:tag="option" value="$auth.prefix"
alt="$auth.description"> <pt:logic.value pt:value="$auth.name"/></pt:core.html>
             </pt:logic.foreach>
          </select>
          </td> </tr>
</pt:logic.iftrue>
<pt:logic.iffalse>
   <!-- Hidden input for single auth source. -->
   <pt:logic.foreach pt:data="authsources" pt:var="auth">
      <pt:core.html pt:tag="input" type="hidden" name="authsource" alt=""
value="$auth.prefix"/>
   </pt:logic.foreach>
</pt:logic.iffalse>
</pt:logic.if>
</pt:logic.iftrue>
<pt:logic.iffalse><!-- Otherwise no auth sources for this resource.
--></pt:logic.iffalse>
</pt:logic.if>
 <tr><!-- Login -->
   <td align="right"></td>
   <td align="left">
   <pt:core.html pt:tag="input" type="submit" value="$#2.ptmsgs_login"/>
   </td>
</tr></table>
</FORM>
<br>
```

```
<SCRIPT language="JavaScript">
function submitform(evt)
{
   evt = (evt) ? evt : event;
   var charCode = (evt.charCode) ? evt.charCode : ((evt.which) ? evt.which :
evt.keyCode);
   if (charCode == 13 || charCode == 3)
   {
   document.loginform.submit();
   }
return true;
}
</SCRIPT
```

The login page can use the `pt:ensemble.loginlink` tag to retrieve the external URL prefix defined for the resource. For example, if the external URL prefix of the resource is http://www.ensemble.com/app/ and the desired page after login is http://www.ensemble.com/app/pages/mainpage.html, then the full login link would be made by adding pages/mainpage.html to the login link prefix as shown in the sample code below.

```
<pt:ensemble.loginlink pt:level="4" pt:key="loginurlprefix"/>
var loginLink = "<pt:logic.value pt:value="$loginurlprefix"/>" +
"pages/mainpage.html";
```

## 3.3 Creating a Custom Oracle WebCenter Ensemble Error Page

A custom error page can be displayed if there is an error in the Oracle WebCenter Ensemble login process.

The `pt:common.error`, `errortext` and `errorcodes` tags allow you to insert Oracle WebCenter Ensemble error information into a custom error page.

**Note:** If these tags are included on a page, errors will no longer be displayed in the normal error location and will not be available after the page has been displayed.

By itself, the `pt:common.errortext` tag displays only the first error message, or the custom error message defined in the pt:text attribute. Other errors, as well as exception stack traces and extended error messages, will be ignored. Combined with the `pt:common.errorcodes` tag and `pt:logic` tags, the `pt:common.errortext` tag can be used to display all error codes in memory. (If the errors have already been displayed, no error codes will be available.)

The example below (errorpage.jsp) illustrates how to retrieve and display a collection of errors and how to replace system errors with a custom error message. This example uses `pt:logic` tags to display the error collection. For details on adaptive tags, see Section 2.2.2, "Adaptive Tags".

```
<%@page contentType="text/html;charset=UTF-8"%>
<HTML> <head> <link rel='stylesheet' type='text/css' href='css/main.css'/> </head>
<BODY>
<SPAN xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<table class="banner" cellpadding="0" cellspacing="0"> <tr>
   <td class="appLogo"></htm:td>
   <td class="liquid"></htm:td>
</tr> </table>
<br/>
<P><pt:logic.value pt:value="$#11.ptmsgs_login"/> </P>
```

```
<P>
<pt:common.errorcode pt:key="errorcodes"/>
<pt:logic.foreach pt:data="errorcodes" pt:var="code">
   <pt:common.errortext/>
   <br>
   <pt:logic.value pt:value="$#12.ptmsgs_login"/><pt:logic.value
pt:value="$code"/>
   <br>

<!-- This is how you would override a specific error with a new message. -->
<!--
<pt:logic.intexpr pt:expr="($code)==2010" pt:key="isrequestedresource"/>
<pt:logic.if pt:expr="$isrequestedresource">
<pt:logic.iftrue>
   <pt:common.errortext pt:text="The requested resource is not in the resource
map. This is a custom error message."/>
</pt:logic.iftrue>
<pt:logic.iffalse>
   <pt:common.errortext/>
</pt:logic.iffalse>
</pt:logic.if>
-->

</pt:logic.foreach>
</P>
<P><pt:logic.value pt:value="$#13.ptmsgs_login"/></P>
</SPAN> </BODY> </HTML>
```

## 3.4 Creating a Custom Oracle WebCenter Ensemble Post-Login Page

The post-login page is an interstitial page that can be used to display messages or gather input from the user after the login form is submitted.

The example below (interstitialpage.jsp) displays a user agreement that requires approval. This page uses adaptive tags to display errors and form elements. For details on adaptive tags, see Section 2.2.2, "Adaptive Tags".

```
<%@page contentType="text/html;charset=UTF-8"%>
<HTML>
<BODY>
<SPAN xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<FORM action="./processinterstitialpage.jsp" method="POST">
<P>
<TABLE>
<TR>
<TD><CENTER><B><pt:logic.value pt:value="$#3.ptmsgs_samples"/></B></CENTER>
</TD></TR>
<TR>
   <TD>
   <TEXTAREA name="thetext" rows="15" cols="80">
   Owner: This web site belongs to Sample Company United States, Inc. Sample
Company may change or terminate this web site or any parts thereof.
   Agreement: Your use of this web site constitutes your agreement with Sample
Company to operate under the auspices of, and to act in
   concordance with, these Terms and Conditions of use.  By clicking "Agree"
below, you are confirming your agreement, which will also be confirmed
   by merely accessing this web site beyond this page.
   Continuing Agreement: Sample Company may modify or alter these Usage Terms at
any time.  Further usage of this web site after the terms have
   been modified confirms your agreement with the modified Usage Terms.
   Use of Materials: Sample Company owns all of the data on this web site or has
secured permission from a third party to use the material. Usage of
   this material in any way outside this web site is strictly prohibited.
   </TEXTAREA>

    <pt:common.error>
    <P>
    <B><FONT color="red"><pt:logic.value pt:value="$#10.ptmsgs_
login"/></FONT>:</B>
    <pt:common.errortext/>
    </P>
    </pt:common.error>

    <P>
    <pt:core.html pt:tag="input" type="checkbox" name="agreement" alt="$#0.ptmsgs_
samples" value="agree"/>
    <pt:logic.value pt:value="$#1.ptmsgs_samples"/> &nbsp;&nbsp;&nbsp;&nbsp;
    <pt:core.html pt:tag="input" type="submit" value="$#2.ptmsgs_samples"/>
    </P>
    </TD>
</TR>
</TABLE></SPAN></BODY></HTML>
```

When the user clicks the submit button, the processing page (processinterstitialpage.jsp) sets the `runner_post_interstitial_complete header` to **true**, which directs the browser to the resource. If the user did not select **Accept** on the post-login page, a message is displayed including a link back to the agreement. This file also provides error text for the error page in case processing fails.

```
<%@page contentType="text/html;charset=UTF-8"%>
<HTML>
<BODY>
<SPAN xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:common.error>
<P>
<B><FONT color="red"><pt:logic.value pt:value="$#10.ptmsgs_login"/></FONT>:</B>
<pt:common.errortext/>
</P>
</pt:common.error>

<%
if (request.getParameter( "agreement" ) != null)
{
   out.println( "<P>Ensemble post-login interstitial page processing login
completion error.
   Contact your system administrator.</P>");
   response.addHeader( "runner_post_interstitial_complete", "true");
   session.invalidate();
}
else
{
    out.println( "<P>If you do not consent to the web site usage agreement, you
will not be able to access any content.</P>");
    out.println( "<P>Click <A href=\"./interstitialpage.jsp\">here</a> to view the
usage agreement again, or ");
    out.println( "you can use <A href=\"http://www.google.com\">Google</a> to find
another web site.</P>");
}
%>
</SPAN></BODY></HTML>
```

## 3.5 Creating a Custom Oracle WebCenter Ensemble Post-Logout Page

The post-logout page is displayed when a user logs out of the resource.

In most cases, this page simply displays a message that informs users that they have been logged out of the system. The `pt:ensemble.ssologout` tag can be added to any proxied page to display a link that logs out of all resources and directs the browser to the post-logout page associated with the user's experience definition. For details on adaptive tags, see Section 2.2.2, "Adaptive Tags".

## 3.6 Configuring Custom Oracle WebCenter Ensemble Login Pages

Custom login resources and pages are configured through the associated experience definition.

To define the login resource, create a resource and select **Is login resource** on the General tab. (If the application server is already registered as a resource in Oracle WebCenter Ensemble, confirm that the login resource setting is enabled.) To deploy custom pages in Oracle WebCenter Ensemble, edit the associated experience definition. On the Log In Settings page, define the login resource and any of the following custom pages:

| Custom Page | Description |
|---|---|
| Pre-login page | Displayed before attempting to authenticate the user. |
| Login page | Displayed only when form authentication is being used; provides the form for login. |
| Post-login page | Displayed to the user after successful authentication and before the resource is accessed. |
| Error page | Displayed if there is an error in the login process. |
| Post-logout page | Displayed after the user logs out of the resource. |

**Note:** The settings in the experience definition are used regardless of the authenticator used to access a resource. If the required authenticator uses a login page and there is no login page configured in the experience definition, the user will be presented with a blank page and will be unable to authenticate. For details, see the *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter Ensemble* Chapter 8, 'Experience Definitions'.

## 3.7 Oracle WebCenter Ensemble Login Headers

Communication between login resource pages and Oracle WebCenter Ensemble is done using HTTP headers

The following table describes the available headers, including how and when they are used. Error and Post-logout pages are considered terminal pages and do not communicate with Oracle WebCenter Ensemble using headers.

*Table 3–1 Oracle WebCenter Ensemble Login Headers*

| Property Type | Property Value | Property Description |
|---|---|---|
| Pre-login | runner_pre_interstitial_complete | true indicates that the pre-login page has completed successfully. Oracle WebCenter Ensemble proceeds to the login page. false (or no header) means the page has not completed successfully. The pre-login page is displayed again. |
| Login | runner_username | The user name used to authenticate the user. |
| Login | runner_password | The password used to authenticate the user. |
| Login | runner_authentication_provider | The provider for authentication. The only valid value is portal. If the header is not present, the provider defaults to portal. |
| Login | runner_portal_authentication_source | The authentication source against which to authenticate the user. This is the same as the authentication source the user would use to log in to Oracle WebCenter Ensemble. |
| Post-login | runner_post_interstitial_complete | true indicates that the post-login page has completed successfully. Oracle WebCenter Ensemble proceeds to the resource. false (or no header) means the page has not completed successfully. The post-login page is displayed again. |

# 4

# Oracle WebCenter Ensemble REST APIs

Oracle WebCenter Ensemble REST APIs provide the following functionality:

- Allow remote web services to retrieve information about resources and pagelets from Oracle WebCenter Ensemble; for details, see Section 4.1, "Data Retrieval APIs".
- Inject pagelets into non-proxied pages, allowing Oracle WebCenter Ensemble to act as a portlet provider for Oracle WebCenter Interaction, Oracle WebLogic Portal, or other third-party portals for details, see Section 4.2, "Pagelet Inject API".

REST stands for Representational State Transfer and is a simple way of providing APIs over HTTP. The basic principles of REST are:

- API URLs point to the resource being used, rather than a generic method endpoint.
- Requests use standard HTTP verbs for simplified CRUD methods. This is a read-only API and allows GET requests only.
- Every request should return a full representation of the object retrieved (pagelet or resource).

## 4.1 Data Retrieval APIs

Two REST APIs are available to retrieve data from Oracle WebCenter Ensemble:

- **Pagelet API**: Allows remote applications to retrieve pagelet data from Oracle WebCenter Ensemble.
- **Resource API**: Allows remote applications to retrieve resource data from Oracle WebCenter Ensemble.

The base URL for all requests is http://<Ensemble base URL>/api/v2/ensemble/

The following arguments are available:

| Argument | Returns | Example: Pagelet API | Example: Resource API |
|---|---|---|---|
| None | All pagelets or resources. | http://myensemble.com/api/v2/ensemble/pagelets | http://myensemble.com/api/v2/ensemble/resources |
| Library or Resource name | All pagelets within a specific library, or a specific resource. | http://myensemble.com/api/v2/ensemble/pagelet/samples/ | http://myensemble.com/api/v2/ensemble/resource/sampleresource/ |

| Argument | Returns | Example: Pagelet API | Example: Resource API |
|---|---|---|---|
| name | The pagelet with the specified name. | http://myensemble.com/api/v2/ensemble/pagelets?name=samplepagelet | To retrieve a specific resource, use the syntax above. |
| resultCount | List of all pagelets or resources, limited to the specified result count. | http://myensemble.com/api/v2/ensemble/pagelets?resultCount=3 | http://myensemble.com/api/v2/ensemble/resources?resultCount=3 |
| id | A pagelet or resource with the specified id. | http://myensemble.com/api/v2/ensemble/pagelets?id=5 | http://myensemble.com/api/v2/ensemble/resources?id=5 |
| owner | A list of resources associated with the specified user. | n/a | http://myensemble.com/api/v2/ensemble/resources?owner={35DE6AF8-ABB6-4e9b-B4E1-D1FC492F2BE8} |
| externalurlprefix | A list of resources associated with the specified URL. | n/a | http://myensemble.com/api/v2/ensemble/resources?externalurlprefix=http://joesmith.amer.bea.com:80/login/ |
| format | The format to use in the response (xml or json). Responses are returned in XML by default. | http://myensemble.com/api/v2/ensemble/pagelets?format=json | http://myensemble.com/api/v2/ensemble/resources?format=json |

For example, the http://myensemble.com/api/v2/ensemble/pagelets?id=5 request returns the following:

```
<ns2:Pagelet>
<name>attributepagelet</name>
<desc>This pagelet uses attributes.</desc>
<id>5</id>
<lastModified>2008-03-03T13:12:44.683-08:00</lastModified>
<created>2008-03-03T13:12:44.683-08:00</created>
<m_strLibraryName>samples</m_strLibraryName>
<parent_resource_name>samples resource</parent_resource_name>
<parent_resource_desc>This resource contains the standard sample pages.</parent_
resource_desc>
<parent_resource_id>8</parent_resource_id>
<parent_resource_external_url_list>/samples/</parent_resource_external_url_list>
-
    <parent_resource_external_url_list>
https://daniilk-w2k3.devnet.plumtree.com:443/samples/
</parent_resource_external_url_list>
-
    <m_strCodeSample>
<pt:ensemble.inject pt:name="samples:attributepagelet"
param1="0"
param2="">

</pt:ensemble.inject>
</m_strCodeSample>
<m_bPublishDocs>true</m_bPublishDocs>
<m_addInlineRefreshToAllUrls>false</m_addInlineRefreshToAllUrls>
```

```
<m_refreshInterval>0</m_refreshInterval>
<url_suffix>attributepagelet.html</url_suffix>
<pagelet_external_url>/inject/v2/pagelet/attributepagelet/samples</pagelet_
external_url>
<parametersTransport>PAGELET_REALM</parametersTransport>
<m_bAllowAll>true</m_bAllowAll>
</ns2:Pagelet>
```

If the format is specified as JSON (http://myensemble.com/api/v2/ensemble/pagelets?id=5&format=json), the following response is returned:

```
{"ns2$Pagelet":{"@xmlns":{"ns2":"http:\/\/social.bea.com\/ensemble"},"name":{"name
":"attributepagelet"},"desc":{"desc":"This pagelet uses
attributes."},"id":{"id":"5"},"lastModified":{"lastModified":"2008-03-03T13:12:44.
683-08:00"},"created":{"created":"2008-03-03T13:12:44.683-08:00"},"m_
strLibraryName":{"m_strLibraryName":"samples"},"parent_resource_name":{"parent_
resource_name":"samples resource"},"parent_resource_desc":{"parent_resource_
desc":"This resource contains the standard sample pages."},"parent_resource_
id":{"parent_resource_id":"8"},"parent_resource_external_url_list":[{"parent_
resource_external_url_list":"\/samples\/"},{"parent_resource_external_url_
list":"https:\/\/daniilk-w2k3.devnet.plumtree.com:443\/samples\/"}],"m_
strCodeSample":{"m_strCodeSample":"<pt:ensemble.inject
pt:name=\"samples:attributepagelet\"\nparam1=\"0\"\nparam2=\"\">\n\n<\/pt:ensemble
.inject>"},"m_bPublishDocs":{"m_bPublishDocs":"true"},"m_
addInlineRefreshToAllUrls":{"m_addInlineRefreshToAllUrls":"false"},"m_
refreshInterval":{"m_refreshInterval":"0"},"url_suffix":{"url_
suffix":"attributepagelet.html"},"pagelet_external_url":{"pagelet_external_
url":"\/inject\/v2\/pagelet\/attributepagelet\/samples"},"parametersTransport":{"p
arametersTransport":"PAGELET_REALM"},"m_bAllowAll":{"m_bAllowAll":"true"}}}
```

## 4.2 Pagelet Inject API

By entering a proxy URL into the portal portlet source code, Oracle WebCenter Ensemble will load up the pagelet as a portlet. The proxy URL must use the following format:

http://host:port/inject/v2/pagelet/libraryname/pageletname?instanceid=55&content-type=html

where libraryname and pageletname refer to the library and pagelet configured in Oracle WebCenter Ensemble.

> **Note:** When using the pagelet inject API as the URL for a Portlet Web Service in Oracle WebCenter Interaction, you must switch "pagelet" to "portlet" in the URL. For example, the above URL would become:
> http://host:port/inject/v2/**portlet**/libraryname/pageletname?instanceid=55&content-type=html

The query string arguments to the above call define how the pagelet is to be returned. The following parameters are defined:

- **instanceid:** Optional. The instance ID of the pagelet.
- **content-type**: The return type. Three types are supported:
  - **javascript**: Returns injectable code.

- **html**: Returns the pagelet markup with its associated PTPortlet object.
- **iframe**: Returns an IFrame that points back to the inject api, filling the IFrame with the pagelet content, instead of directly inline with the page. The IFrame can be styled by providing a set of query string parameters.

| Parameter | Description | Default |
|---|---|---|
| ifwidth | Sets the width of the IFrame; can be specified in percent '%' or pixels 'px', for example: ifwidth=500px. Can be set to 'auto' to automatically resize the IFrame to fit the content within. For details, see Section 4.2.1, "Using Automatic Resizing with IFrames". | 100% |
| ifheight | Sets the height of the IFrame; can be specified in percent '%' or pixels 'px', for example: ifheight=500px. Can be set to 'auto' to automatically resize the IFrame to fit the content within. For details, see Section 4.2.1, "Using Automatic Resizing with IFrames". | No default |
| ifborder | Sets the border of the IFrame. | 'none' |
| ifalign | Sets the align rule within the IFrame, for example: ifalign=center. | No default |
| ifdesc | Sets the description of the IFrame. | No default |
| ifmarginheight | Sets the margin height; can be specified in percent '%' or pixels 'px', for example: ifmarginheight=500px. | No default |
| ifmarginwidth | Sets the margin width; can be specified in percent '%' or pixels 'px', for example: ifmarginwidth=500px. | No default |
| ifscrolling | Sets the scrollbars of the IFrame. Accepted values: yes/no/auto. | auto |
| ifstyle | Sets the CSS style of the IFrame | No default |
| ifclass | Sets the CSS class of the IFrame. | No default |

- **csapi**: Sets whether the CSAPI will be included with the pagelet response (true or false). Including the CSAPI is optional, but the pagelet included in the response relies on the CSAPI libraries being present on the page where the pagelet is to be rendered. If csapi=false, then the CSAPI libraries must be included with the parent page (usually in the HEAD section).
- **onhtttperror:**: When a pagelet request results in a 403, 404 or any other error code, Oracle WebCenter Ensemble can forward the error code and the error page itself to the browser for display to the user. The onhttperror parameter accepts the following values:
  - comment (default): Oracle WebCenter Ensemble will create an HTML comment in place of the failing pagelet (the failing pagelet will simply not be displayed).
  - inline: The pagelet error along with the server error page will be displayed inline where the pagelet would normally be shown on the page.
  - fullpage: The http error will consume the whole page. This mode is only available if Oracle WebCenter Ensemble controls the parent page.

For example, the following URL points to the linkpagelet in the samples library:

http://proxy:port/inject/v2/pagelet/samples/linkspagelet?content-type=iframe&csapi=true&ifheight=123px&ifclass=myclass

This URL should result in markup similar to the code below.

---

**Note:** The IFrame source points back to the inject API, but this time the content-type parameter is set to html. This feature adds an additional step in the pagelet retrieval. The csapi parameter is seet to true on the subsequent call to get the IFrame contents so that the required CSAPI content is included in the IFrame (if this was not the case, javascript resolve errors would be returned because the pagelet code cannot access any CSAPI script included outside the IFrame).

---

```
<html>
 <head>
 </head>
 <body>
  <iframe frameborder="none" class="myclass" width="100%" height="123px"
scrolling="auto"
src="http://proxy:port/inject/v2/pagelet/samples/linkspagelet?asdg=asdfgas&param=t
rue&content-type=html&jswrap=false&csapi=true">
   <html>
    <head>
     <script
src="http://proxy:loginserverport/loginserver/ensemblestatic/imageserver/plumtree/
common/private/js/jsutil/LATEST/PTUtil.js" type="text/javascript"> </script>
     <script
src="http://proxy:loginserverport/loginserver/ensemblestatic/imageserver/plumtree/
common/private/js/jsutil/LATEST/PTDateFormats.js" type="text/javascript"></script>
     <script
src="http://proxy:loginserverport/loginserver/ensemblestatic/imageserver/plumtree/
common/private/js/jsxml/LATEST/PTXML.js" type="text/javascript"></script>
     <script
src="http://proxy:loginserverport/loginserver/ensemblestatic/imageserver/plumtree/
common/private/js/jsportlet/LATEST/PTPortletServices.js"
type="text/javascript"></script>
    </head>

    <body>
     <div id="pt-pagelet-content-1" class="pagelet-container" style="display:
inline;">
      <span xmlns:pt="http://www.plumtree.com/xmlschemas/ptui/">
      Pagelet links:
      <br/>
      <a href="http://proxy:port/inject/RP_PID393219_I1/headpagelet1.html">The
first pagelet</a>
      <br/>
      <a href="http://proxy:port/inject/RP_PID393219_I1/headpagelet2.html">The
second pagelet</a>
      <br/>
      <a href="http://proxy:port/inject/RP_PID393219_I1/csapipagelet.html">The
csapi pagelet</a>
      <br/>
      <a href="http://proxy:port/inject/RP_PID393219_I1/linkspagelet.html">This
pagelet</a>
      <br/>
      </span>
     </div>
    </body>
   </html>
  </iframe>
 </body>
</html>
```

### 4.2.1 Using Automatic Resizing with IFrames

The Oracle WebCenter Ensemble pagelet inject API can automatically resize the IFrame that encapsulates pagelet content. The resizing is done so that the IFrame stretches to fit the content within. To use this feature, the ifwidth and ifheight parameters must be set to 'auto' as shown in the example below:

http://proxy:port/inject/v2/pagelet/samples/linkspagelet?content-type=iframe&csapi=true&ifheight=auto&ifwidth=auto&ifclass=myclass

In addition, this feature relies on an external page on the same domain as the consumer page. This page is included into the pagelet IFrame as an internal hidden IFrame. This page collects the sizing information and passes it on to the parent consumer page. This page must be deployed in the same directory as the consumer page. An example is shown below.

```
<html>
  <head>
    <title>Resizing Page</title>
    <script type="text/javascript">
function onLoad() {
var params = window.location.search.substring( 1 ).split( '&' );
var height;
var width;
var iframe;

for( var i = 0, l = params.length; i < l; ++i ) {
var parts = params[i].split( '=' );
switch( parts[0] ) {
case 'height':
height = parseInt( parts[1] );
break;
case 'width':
width = parseInt( parts[1] );
break;
case 'iframe':
iframe = parts[1];
break;
}
}
window.top.updateIFrame( iframe, height, width );
}

if (window.addEventListener) {
window.addEventListener("load", onLoad, false)
} else if (window.attachEvent) {
window.detachEvent("onload", onLoad)
window.attachEvent("onload", onLoad)
} else {
window.onload=onLoad
}
    </script>
  </head>
  <body>
  </body>
</html>
```

# 5

# Oracle WebCenter Ensemble API Libraries

This page lists API libraries for use in Oracle WebCenter Ensemble development. All documentation can be found on the Oracle Technology Network at `http://www.oracle.com/technology/index.html`.

## 5.1 Oracle WebCenter Interaction Development Kit (IDK)

These API libraries provide detailed documentation on IDK objects and methods. To access documentation for previous versions or download the entire documentation package, download the appropriate version of the IDK. For details on using these APIs, see Section 2.1, "Oracle WebCenter Interaction Development Kit (IDK) Proxy API."

## 5.2 Oracle WebCenter Interaction Scripting Framework

The Oracle WebCenter Interaction Scripting Framework is a collection of client-side JavaScript libraries that provide services to pagelets and hosted proxied pages. Most services are provided by the JSPortlet API. Opener functionality is provided by the Common Opener API. For lower-level functionality, see the JSXML package. For details on using these APIs, see Section 2.2.1, "Adaptive Pagelet Design Patterns."

## 5.3 Adaptive Tags

Adaptive Tags are XML tags that can be included in the markup returned by any proxied page, including pagelets. These tags provide access to key portal components and support advanced attribute replacement. For details on using tags, see Section 2.2.2, "Adaptive Tags".

A separate set of classes are used to create custom Adaptive Tags for use in pagelets and proxied pages. For details on creating custom tags, see Section 2.2.2.8, "Creating Custom Adaptive Tags".

# 6

# Additional Development References

The following references provide additional information for use in Oracle WebCenter Ensemble development. All documentation can be found on the Oracle Technology Network at `http://www.oracle.com/technology/index.html`.

**CSP**

CSP is a platform-independent protocol based on the open standard of HTTP 1.1. The syntax of communication between the portal and remote servers is defined by CSP. CSP defines custom headers and outlines how Oracle WebCenter Interaction and Oracle WebCenter Ensemble services use HTTP to communicate and modify settings. The Oracle WebCenter Interaction Development Kit (IDK) provides simplified, stable interfaces that allow you to write code that communicates using CSP. The current version of CSP is 1.4.

**Oracle WebCenter Application Accelerator for Microsoft .NET**

The Oracle WebCenter Application Accelerator for Microsoft .NET is a collection of libraries and Visual Studio 2005 integration features that supporteasy authoring of ASP.NET 2.0 and WSRP portlets. The Oracle WebCenter Application Accelerator for Microsoft .NET includes the Oracle WebCenter PortletToolkit for .NET. Portlets can be authored for both Oracle WebCenter Interaction and Oracle WebLogic Portal. Development guides are available for both environments.

**Oracle WebCenter Analytics APIs**

Oracle WebCenter Analytics delivers comprehensive reporting onactivity and content usage within portals and composite applications, allowing you to know and meet user information needs. The OpenUsage and Query APIs provide access the Oracle WebCenter Analytics functionality from customapplications.

- The OpenUsage API allows you to to raise Oracle WebCenter Analytics events from custom portlets and applications and store them in the database.
- The Query API allows you to query data in the Oracle WebCenter Analytics database.

**Oracle WebCenter JSR-168 Container**

The Oracle WebCenter JSR-168 Container is an implementation of the JSR-168 JCP standard for portlet authoring.