Oracle® WebLogic JSR-170 Adapter

Developer's Guide

10*g* Release 3 (10.3)
September 2008

Oracle WebLogic JSR-170 Adapter Developer's Guide, 10*g* Release 3 (10.3)

Contents

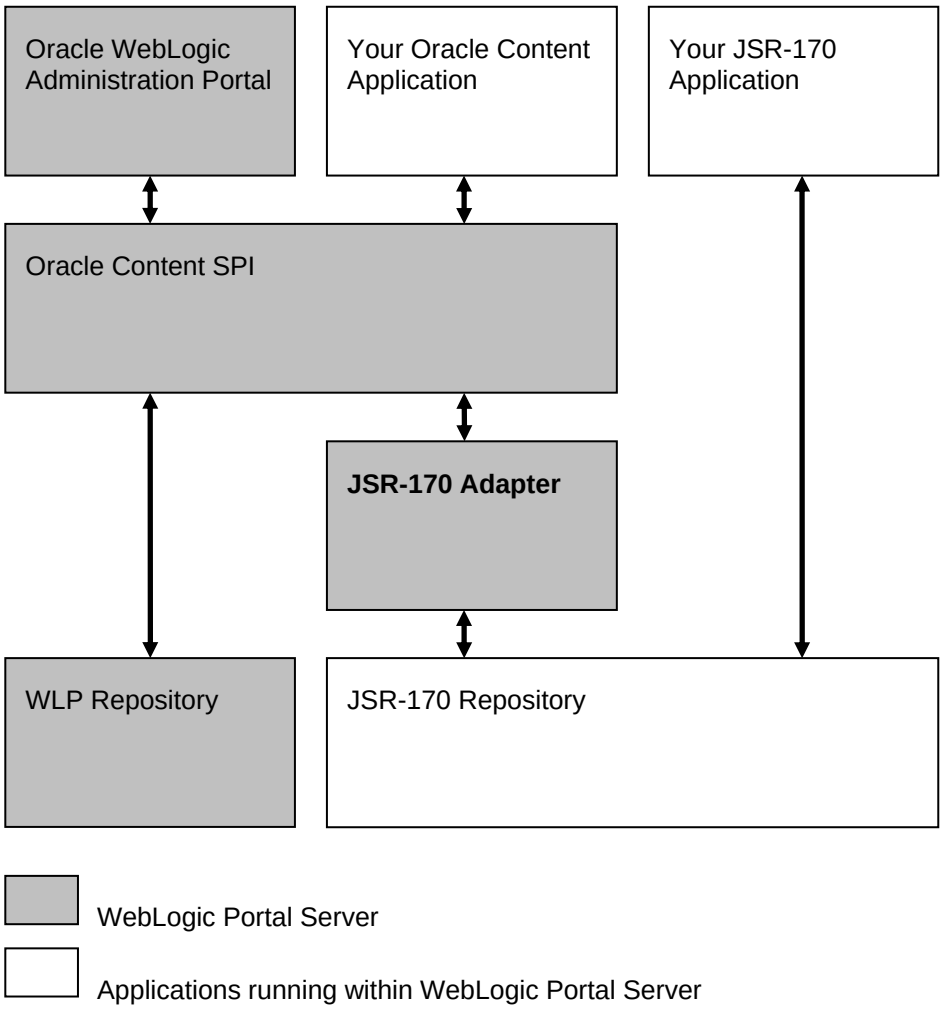
Overview

The JSR-170 Adapter is a feature of Oracle WebLogic Portal 9.x that provides access through the WebLogic Portal content service provider interface (SPI) to any JSR-170-compliant repository. This lets applications written to the WebLogic Portal content SPI access the JSR-170 repository as well as allowing some repository administration functions to be performed through the WebLogic Administration Portal.

This document shows you how to connect WebLogic Portal to a JSR-170-compliant content repository. It also describes the basic repository functions available through the WebLogic Administration Portal and the WebLogic Portal content SPI and how those are mapped to the JSR-170 level.

Architecture

The JSR-170 Adapter serves as bridge between the WebLogic Portal content SPI and a JSR-170-compliant repository running either within WebLogic Server or externally (such as in a separate JVM, either on the same machine or on a remote machine). The relationships between the various elements is illustrated in the following diagram:



Getting Started

Select a JSR-170 repository implementation

The first step to integrating a JSR-170 repository with WebLogic Portal is to select a repository implementation.

A list of JSR-170 implementations with their TCK test results is available at www.day.com.

Two repositories currently available are **Apache Jackrabbit** (http://incubator.apache.org/jackrabbit/) and **Day CRX** (http://www.day.com/site/en/index/products/content-centric_infrastructure/content_repository.html).

Deploy the JSR-170 repository

The JSR-170 Adapter supports two mechanisms for connecting to the repository: Repositories running within the same JVM as the server are accessed through JNDI. Repositories running in another JVM (possibly on a remote machine) are accessed through RMI.

For information on deploying your repository, consult the application server documentation, as well as that of the repository vendor. If you are deploying the repository on WebLogic Server, deploy your repository implementation just like any other WebLogic application.

Configure WebLogic Portal to access the JSR-170 repository

Once the JSR-170 repository is deployed and running it must be connected to WebLogic Portal so that it appears as part of its Virtual Content Repository. To add the JSR-170 repository:

1. Open the WebLogic administration portal.
2. In the Content Management tool, select the Repository page to view the repositories.
3. Click **Virtual Content Repository** at the top of the **Repository List** tree.
4. In the **Editor** pane to the right, click the **Add Repository Connection** button.
5. Fill-in the following information:

Field	Entry
Name	A name you choose
Connection Class	To access an in-process repository: com.day.content.spi.jsr170.JNDIRepository To access an external repository: com.day.content.spi.jsr170.RMIRepository
Username	Your WebLogic user name
Password	Your WebLogic password

Field	Entry
Enable Library Services	false

6. Now you must add custom properties specific to the JSR-170 Adapter.

   The properties needed depend on whether you are using the **JNDIRepository** connection class or the **RMIRepository** connection class (see point 6, above):

   Click the **Add Property** button for each property you want to add, and fill-in the key-value pairs as follows:

   For the **JNDIRepository** connection class:

Key	Value
jsr170.workspace	The name of the workspace in the JSR-170 repository you want to connect to. Most JSR-170 repositories have a default workspace called default. Note that in Oracle terms, each JSR-170 workspace becomes a separate "repository" within the larger WLP Virtual Content Repository.
jsr170.jndi.name	The JNDI name assigned to the JSR-170 repository web application in its web.xml configuration file. Most JSR-170 repositories have a default JNDI name of repository.

   For the RMIRepository connection class:

Key	Value
jsr170.workspace	The name of the workspace in the JSR-170 repository you want to connect to. Most JSR-170 repositories have a default workspace called default. Note that in Oracle terms, each JSR-170 workspace becomes a separate "repository" within the larger WLP Virtual Content Repository.
jsr170.rmi.hostname	The name of the host machine on which the RMI-enabled JSR-170 repository is running. The default is localhost.
jsr170.rmi.port	The port on the host machine through which the JSR-170 repository is accessed. The default is 1099.
jsr170.rmi.boundname	The RMI bound name of the JSR-170 repository. The default is repository.

7. You can leave the **Repository Cache Settings** as is.

8. Click **Create**. The new repository should appear in the tree on the left hand side.

Working with JSR-170 within WebLogic Portal

The sections below describe the main reading, writing and administration methods of the WebLogic Portal content SPI and how the same functions are supported through the WebLogic Administration Portal, through an example commercial JSR-170 repository (Day CRX, see jcr.day.com) and (where applicable) through the example JSR-170 portlet.

For each WebLogic SPI method, the effect on the underlying JSR-170 repository is described and the equivalent JSR-170 methods are explained. Where mismatches between the two content models cause special behavior, this is explained as well.

Note: For a clearer understanding of how the WebLogic Portal content methods affect the JSR-170 repository, use the Day CRX repository, which provides a graphical Content Explorer interface through which changes to the JSR-170 repository made through the WebLogic Portal content SPI can be viewed in real time from the JSR-170 viewpoint. See http://jcr.day.com.

The following section does not detail every method in the WebLogic Portal content SPI (and its JSR-170 equivalent), but does cover the most important ones. For more details consult the WebLogic Portal and JSR-170 Javadocs. The relevant interfaces in WebLogic Portal are found in the package **com.bea.content.spi**. In JSR-170 they are in the package **javax.jcr**. Note that WebLogic Portal also provides convenience methods in **com.bea.content.Node** and **com.bea.content.Property**. Since these methods simply delegate to **NodeOps**, only the actual core **NodeOps** methods are discussed. For information on the corresponding convenience classes, consult the WebLogic Portal Javadoc.

Accessing Nodes

WebLogic Portal Method in `com.bea.content.spi`	**JSR-170 Method in `javax.jcr`**
NodeOps.getNode(ID nodeId)	Session.getNodeByUUID(String uuid)
NodeOps.getNode(String path)	Session.getItem(String absPath) Returns an Item which must be cast to Node. Alternatively, having already acquired a Node, one can call Node.getNode(String relPath) to get a descendant Node.
NodeOps.getNodeChildren(ID parentId, int type)	Session.getNodeByUUID(String uuid) followed by Node.getNodes() on the returned Node.
NodeOps.getNodes(ID[] nodeIds)	Multiple calls to Session.getNodeByUUID(String uuid).

Accessing Nodes through the WebLogic Administration Portal is done by first clicking the **Content** button in at the top of the page and then selecting **Content** in the drop-down

menu above the tree view. Then one simply navigates through the tree and clicking on the appropriate node.

Similarly when using Day CRX to access the JSR-170 repository directly, node access is done by first entering the Content Explorer, then navigating through the tree and clicking on the desired node tree.

In the example JSR-170 portlet, navigate the tree structure to access nodes.

Reading Properties

WebLogic Portal Method in *com.bea.content.spi*	**JSR-170 Method in *javax.jcr***
NodeOps.getProperties(ID nodeId)	Session.getNodeByUUID(String nodeId) followed by Node.getProperties() on returned Node.
NodeOps.getPropertyBytes(ID propertyId)	Session.getItem(String absPath) Returns an Item which must be cast to Property, then read with Property.getStream().

Reading properties through both the WebLogic Administration Portal, CRX Content Explorer and the example JSR-170 portlet is also done through tree navigation.

Adding Nodes

WebLogic Portal Method in *com.bea.content.spi*	**JSR-170 Method in *javax.jcr***
NodeOps.createNode(ID parentId, String newNodeName, int type)	After getting the parent node (see above) call Node.addNode(String name).
NodeOps.createNode(ID parentId, String newNodeName, int type, ID objectClassId, Property[] properties)	After getting the parent node (see above) call Node.addNode(String name, String nodeTypeName) then add the desired properties with Node.setProperty(String name, Value value)

To add a node through the WebLogic Administration Portal navigate to the desired parent node and then click **Add Content** in the blue area above the Edit Pane. This will bring up a field where you can type in a name for the node. Then click **Choose Type** and select an

object class (actually a JSR-170 node type). Keep in mind that node types with mandatory child nodes cannot be created through this interface (or through the SPI) so many of the node types built into the underlying JSR-170 implementations will cause an error if chosen.

To add a node through the CRX Content Explorer navigate through the tree to the desired parent node and click it. Click the "folder symbol in the top menu and select **New Node**. In the upper right pane type the name of the new node and select its node type (note that through direct access to the JSR-170 repository, all node types are available). Now you can add subnodes and properties. You *must* add those subnodes and properties that are required by the node type of the new node. Once your new node is properly populated with subelements click **Save All** in the top menu to persist the changes.

Adding Properties

WebLogic Portal Method in *com.bea.content.spi*	**JSR-170 Method in *javax.jcr***
NodeOps.addNodeContent(ID nodeId, ID objectClassId, Property[] properties)	After getting the parent node (see above) add the desired properties with Node.setProperty(String name, Value value)
NodeOps.updateProperties(ID nodeId, Property[] properties)	After getting the parent node (see above) add the desired properties with Node.setProperty(String name, Value value)

Unlike JSR-170, adding properties one-by-one is not done in WebLogic, instead a node's object class fully defines the set of properties that a node has, and clicking on that node in the Administration portal will reveal a table of those properties where you can edit the values. Adding properties a node in WebLogic is essentially equivalent to assigning an object class to that node. See the next section for information about how to do that.

In the CRX Content Explorer clicking on a node will similarly reveal an edit table with properties defined by the node type already listed (though if they don not actually exist yet, they will be grayed-out). These can be edited, the check-mark button clicked and the change persisted by clicking **Save All**. In addition, if the node type of the node in question allows unstructured properties (also called residual properties in JSR-170 terminology) these can be added as well in the edit pane, with the user choosing the names.

Assigning Node Types/Object Classes

WebLogic Portal Method in *com.bea.content.spi*	**JSR-170 Method in *javax.jcr***

NodeOps.addNodeContent(ID nodeId, ID objectClassId, Property[] properties)	The WebLogic Portal method allows the changing of a object class of an existing node. In JSR-170 the primary node type is assigned at node creation time with Node.addNode(String name, String nodeTypeName).

Assigning a node type in through the WebLogic Administration Portal is one of the steps in the creation of new node (see above), though unlike JSR-170, you can also remove and change the object class of a node later in its life.

In the CRX Content Explorer choosing a node type is done in the drop down menu that appears in the top right pane during the process of adding a node.

Search

Searching through the WebLogic SPI is done using the **com.bea.content.spi.SearchOps** interface. For a description of the syntax of the WebLogic search syntax consult the javadoc for **com.bea.content.expression.ExpressionHelper**.

Searching in JSR-170 is done through **javax.jcr.query.QueryManager** (again, see the Javadoc for details). The syntax of the two available query languages (SQL and XPath) is described in the JSR-170 specification.

In the CRX Content Explorer, search is accessed through the "magnifying glass" button in the top menu. This opens a window where queries in either of the two supported languages can be entered.

Alternatively, the example JSR-170 portlet includes a search function that accepts SQL queries.

Mapping Between Content Models

The WebLogic Portal Content SPI and the JSR-170 API are based on different models. For this reason, not all JSR-170 repository functionality is exposed through the WebLogic SPI and not all Weblogic Portal functionality is available in the underlying JSR-170 repository. The following section outlines the similarities and differences between the two models and how the adapter maps these models to one another.

Workspaces

In JSR-170 a repository may consist of one or more *workspaces*, each of which contains a hierarchy of content items. In WebLogic terminology, a single hierarchy of content items is referred to as a *repository*. So, a WebLogic repository (one entry among possibly many within the larger Virtual Content Repository) corresponds to a JSR-170 workspace. To connect more than one workspace from a single JSR-170 repository to WebLogic, each workspace must be added as a distinct WebLogic repository (it is for this reason that the workspace name property is required in the configuration, see point#6 in section 1.4.3).

Nodes

In both the WebLogic Portal content SPI and JSR-170 a **node** is structural element in a hierarchy at the API level. At the user interface level in WebLogic Portal, a content directory is called a "folder" and a piece of content is called "content" or "content item."

Node Types and Object Classes

JSR-170 **node types** are mapped to WebLogic Portal **object classes**. However, there are two significant differences between the concepts:

1. A JSR-170 node type defines the properties and child nodes a particular node may (or must) have. A WebLogic object class, governs *only the properties* a node may (or must) have. Because of this, the methods of **com.bea.content.ObjectClass** and **com.bea.content.PropertyDefinition** does not expose any additional constraints on child nodes that their underlying JSR-170 node type may define. Nonetheless, the native constraints of the node type (both property and child node) are still enforced by the underlying JSR-170 repository even though these constraints are not visible through the WebLogic SPI.
2. In WebLogic Portal, a node may have no object class. In JSR-170 a node always has a node type. This constraint is still enforced when a JSR-170 repository is accessed though the WebLogic SPI. Consequently, in WebLogic terms, adding a node without an object class is not possible.

In WebLogic, there is also a separate concept of "node type" distinct from that of object class. There are two such WebLogic node types: content and hierarchy. The combining rules between them are simple: a hierarchy node can have child nodes of either type, while a content node can only have child nodes of type content. Both types of nodes can be assigned object classes and have properties. The JSR-170 adapter maps all JSR-170 nodes to WebLogic content nodes. This means that to add a node to the JSR-170 repository through either the WebLogic Administration Portal or the WebLogic SPI, you must add a WebLogic content node. An attempt to add a WebLogic hierarchy node will fail.

Mixin Node Types

JSR-170 distinguishes two types of node types: primary and mixin. Primary node types are the type discussed above. They are assigned upon creation of a node and cannot be changed or removed, short of removing the node itself and creating a new one. Mixin types are node types that specify additional properties to an already-existing node. They can be added at any time in the node's life cycle.

WebLogic Portal object classes are in some ways more similar to mixin types in that nodes can have no object class at all and object classes can be added, removed, or changed during the life of the node.

Because the WebLogic SPI exposes JSR-170 primary node types as WebLogic object classes, the immutable nature of those node types restricts the operations that can be done on their corresponding object class. In particular, once an object class reflecting an underlying JSR-170 node type is assigned, it cannot be removed.

Unstructured Content

In a JSR-170 repository a node may be assigned a node type that permits it to have any number of properties and/or child nodes with any names and of any types.

In WebLogic Portal a node can have (subject to the hierarchy/content combination rule mentioned above) any number of child nodes with any names. However, WebLogic Portal requires that all the properties of a node be defined by name in its object class. As a result, the WebLogic Portal content SPI does not currently support unstructured sets of properties. For example, this means that through the WebLogic SPI, the JSR-170 node type nt:unstructured is not fully supported (its unstructured child node aspect is, but not its unstructured property aspect).

Properties

In both models properties are name-value pairs attached to nodes that store the actual data in the repository.

Property Types

Both models support the property types **binary**, **boolean**, **calendar**/**date**, **double**, **long**, **string** and **undefined**. JSR-170 also supports the types **reference**, **name** and **path**.

Transient Space

In a JSR-170 repository, changes made to a node (adding or deleting child nodes; adding, deleting, or changing the values of properties) are not immediately persisted. Instead, the client of the API makes the required changes and once they are complete, calls save. At this point all the changes are validated against node type restrictions and (if valid) persisted. This arrangement system allows the state of a node to be temporarily invalid while it is "being worked on," and only subject to validation once the "work" is finished.

In contrast, WebLogic Portal persists changes immediately upon the change method being called. Consequently, if an object class specifies that a node must have a particular set of properties and that object class is assigned to a node, all of its required properties must be added to the node at the same time. WebLogic supports this through methods like **com.bea.content.spi.NodeOps.addNodeContent** (see the WebLogic Portal Javadoc for details).

At the JSR-170 level, the adapter deals with this by implicitly performing a save for each WebLogic SPI call that alters nodes or properties. As a result, nodes in JSR-170 that require one

or more *child nodes* (as opposed to properties) cannot be created through the WebLogic Portal content SPI. However nodes whose required child set consists only of properties can be created.

Namespaces

To support namespaces in WebLogic Portal, the adapter allows the client to use namespace prefixes with in the names of nodes, properties, and node types. The only prerequisite is that these prefixes must be mapped to URIs in the namespace registry of the JSR-170 repository underlying the WebLogic repository. Consult the documentation of your JSR-170 implementation for more details on setting namespace mappings. In Day CRX, for example, namespace mappings can be added in the Node Type Administration Console.