

**Creating Your First WebCenter Application
with Java Content Repository**

February 2008

This technical note briefly describes the specification for accessing content repositories in a uniform manner, and then quickly drills down to a simple worked example that accesses a file system. Code is provided for implementing a custom JavaBean data control that you can reuse in your ADF/WebCenter applications.

Tech Note Content[1. Overview](#)[2. Creating the "Hello World" of Content Applications](#)[3. Retrieving the Properties for a Given Item](#)[4. Summary](#)**1. Overview**

Unstructured (“document-based”) information is a critical component of an organization’s overall intellectual capital. Integrating this content into the daily tasks of knowledge workers is a primary goal for Oracle WebCenter.

What is JCR ?

The Java Content Repository API, or JCR for short, was introduced to allow backend-independent access to content, regardless of the underlying repository or the nature of the content (documents, relational content, and so on) .

The JCR 1.0 API, as defined in JSR 170, provides a set of basic capabilities for reading, writing, browsing, and searching content.

How does JCR work ?

JCR describes a content repository as a tree of related items, in which each item can be of a different type. An item is either a node or a property—a node can have child nodes and properties, but a property can only have a value. So the containership hierarchy and the content properties are modeled similarly, the only differentiator being the item type. Your application can determine the type of a particular item and handle it accordingly.

There are predefined node types in JCR, but additional node types may be derived from the basic types to tailor the experience to a particular repository. Of course, the more you tailor the system, the more flexible your application code must be to allow for swapping repositories; a fundamental goal of JCR is to facilitate the repository *independent* coding of applications. JCR also defines a set of property types, for example String, Date, and Binary, but unlike the node types, only predefined property types are allowed; a vendor cannot create additional properties.

JCR describes two basic node types to represent containers (`nt:folder`) and documents (`nt:file`). These are derived from `nt:hierarchyNode` which itself is a child of `nt:base`, the supertype.

What is a JCR data control?

A data control is a container for all the data objects, collections, methods, and operations used to create User Interface (UI) components within your WebCenter application. Each type of data control contains common methods and parameters to publish as links, tables, files, and folders, and to add search and advanced search capabilities for your content. Content repository data controls (also called JCR data controls) enable you to connect to and read from the file system, Oracle Portal, Oracle Content Server, Microsoft SharePoint, and EMC Documentum, as well as JCR 1.0 repositories. You can use JCR data controls to publish content from any JCR repository.

2. Creating the “Hello World” of Content Applications

Let’s start by creating a JCR data control (see the [Integrating Content](#) chapter of the *Oracle WebCenter Framework Developers Guide* for detailed information about data controls). A content repository data control provides access to the JCR APIs of the adapter instance it uses, so you can utilize the full power of JCR without having to know the details of connection and authorization.

The managed bean that you will write to implement the data control’s logic retrieves the JCR session first, and then performs some basic browsing of a repository.

Retrieve the JCR session for a given data control

In order to get access to the JCR session, you need to retrieve the ADF context, look up the connection to the repository, and then perform the login operation, as illustrated in the code below.

```
ADFContext adfCtx = ADFContext.getCurrent();
Context conCtx = adfCtx.getConnectionsContext();
Repository rep = (Repository) conCtx.lookup("myFileDataControl");
Session session = rep.login();
```

The example above uses the plain login method. For the file system adapter, no credentials are necessary. However, there are additional methods that allow you to pass credentials. For repositories such as Content Database, you can configure the login() method to perform a JAAS-based authentication based on the JAAS context; by default, the shared credentials you provided when defining the connection in the wizard will be used.

Browse the nodes of a level in the hierarchy

After creating a connection and obtaining the session, you can perform some basic browsing. For this task, perform the getItem operation, passing the location of the repository that you want to browse as a parameter.

The getItem operation returns the node representing the location, including all its children. The children are either other nodes (nt:file, nt:folder or related node-types or properties). Use the getNodes() method to get an iterator for all the nodes, and getProperties() to get an iterator for all the properties of the given location.

Each node itself has a getNodes() and a getProperties() method, which allow you to walk down the hierarchy tree, as in the example below.

```
try {
    Node node = (Node) session.getItem("/");
    NodeIterator iter = node.getNodes();
    while (iter.hasNext()) {
        Node child = iter.nextNode();
        al.add(child.getName());
    }
}
```

```
    } finally {  
        session.logout();  
    }  
}
```

Create and use the data control

For your reference, here is the complete source code for your first JCR custom bean, which includes the two code snippets you've already seen.

```
package model;  
import java.util.ArrayList;  
import java.util.List;  
import javax.jcr.Node;  
import javax.jcr.NodeIterator;  
import javax.jcr.Repository;  
import javax.jcr.Session;  
import javax.naming.Context;  
import oracle.adf.share.ADFContext;  
  
public class JcrClient {  
  
    public List<String> jcrNodes(String sPath){  
        ArrayList<String> al = new ArrayList<String>();  
  
        try {  
            ADFContext adfCtxt = ADFContext.getCurrent();  
            Context conCtxt = adfCtxt.getConnectionsContext();  
            Repository rep =  
(Repository) conCtxt.lookup("myFileDataControl");  
            Session session = rep.login();  
  
            try {  
                Node node = (Node) session.getItem(sPath);  
                NodeIterator iter = node.getNodes();  
                while (iter.hasNext()) {  
                    Node child = iter.nextNode();  
                    al.add(child.getName());  
                }  
            } finally {  
                session.logout();  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        return al;  
    }  
}
```

After you create the bean, right-click the bean in the Application Navigator and select Create Data Control from the context menu. Congratulations, you have created your first custom content data control!

Now create a new JSPX page, expand the Data Control palette, and expand the `jcrNodes` method. Next, drag Results onto the page and drop it as an ADF Table component. When prompted for the value of `sPath`, use `"/` or any other valid path within the tree you selected when you created the file system data control.

3. Retrieving Properties for a Given Item

The power of content management lies not only in the content stored, but also in the categorization of that content. Metadata identifies pieces of content in a structured manner, allowing easy search and retrieval of the content.

The node types and properties that are available depend upon the model of the specific repository. A document, for example, is represented by a node of type `nt:file`. This node can have properties such as `jcr:created` and `jcr:primaryType`. Also, an `nt:file` node has a child called `jcr:content`, which represents the content of that particular document. In addition, the node can have properties associated with it.

Get the properties of a node

To loop through the properties of a node, call `getProperties()` on the respective node to retrieve a `PropertyIterator`, as illustrated below.

```
pi = child.getProperties();
System.out.println("*** "+child.getName());
while (pi.hasNext()) {
    Property property = pi.nextProperty();
    System.out.println("    "+property.getName()+": "
        +property.getValue().getString());
}
```

As we mentioned earlier, the `jcr:content` node, which is a child of your document node, can have its own properties. To access these properties, simply retrieve the child named `jcr:content` and perform the same operations as above.

```
Node contentNode = child.getNode("jcr:content");
PropertyIterator contentPi = contentNode.getProperties();
while (contentPi.hasNext()) {
    System.out.println("    -"+ contentPi.nextProperty().getName());
}
```

If you want to expose a preset list of attributes in your data control, you can address individual properties using the `getProperty()` method. You provide the path to a particular property and the method retrieves the property object.

To use the `getProperty()` method, you need to understand how the repository exposes its content through JCR. In this particular example, `jcr:primaryType` is a child of the main node, while `jcr:modifiedDate` is a child of the `jcr:content` node. So, to retrieve `jcr:lastModified`, you have to specify the relevant path, specifically, `jcr:content/jcr:lastModified`.

The same property can be repeated in the hierarchy. The `jcr:content` node, for example, also has a `jcr:primaryType` property, which describes its particular type.

Expose document properties using a data control

You have just seen how to use JCR to access content and its properties. Now let's expose this functionality through the data control. You will create a class that represents the document and its properties. To simplify this task, you will use one list to display both the node properties and the `jcr:content` properties.

First, create two helper classes to store the information that will be retrieved from the content repository—one to store the JCR property information and one to represent the document. Each document allows for an array of properties to be stored. The helper classes are shown below.

```
// JCR PROPERTY CLASS
package model;

public class JcrProperty {
    String sName;
    String sValue;

    public String getPropertyName ()
    {
        return sName;
    }

    public void setPropertyName (String propertyName) {
        sName = propertyName;
        System.out.print ("Setting "+propertyName+" = ");
    }

    public String getPropertyValue ()
    {
        return sValue;
    }

    public void setPropertyValue (String propertyValue) {
        sValue = propertyValue;
        System.out.println (propertyValue);
    }
}

// JCR DOCUMENT CLASS
package model;

import java.util.ArrayList;

public class Document {
    String name;
    ArrayList<JcrProperty> jcrProperties;

    public Document () {
        jcrProperties = new ArrayList<JcrProperty> ();
    }

    public void addProperty (String sName, String sValue) {
        JcrProperty jProperty;
        jProperty = new JcrProperty ();
        jProperty.setPropertyName (sName);
        jProperty.setPropertyValue (sValue);
        jcrProperties.add (jProperty);
    }

    public ArrayList<JcrProperty> getProperties () {
        return jcrProperties;
    }
}
```

```

public void setName (String sName) {
    name = sName;
}

public String getName () {
    return name;
}
}

```

Next, modify the JcrClient bean to fill the array objects with the documents returned from the JCR repository.

```

// JCR CLIENT BEAN
public class JcrClient {
    /**
     * Node and property names we're interested in.
     */

    ArrayList<Document> docs = new ArrayList<Document>();

    public ArrayList<Document> jcrNodes (String sPath) {

        try {
            ADFContext adfCtxt = ADFContext.getCurrent();
            Context conCtxt = adfCtxt.getConnectionsContext();
            Repository rep =
                (Repository) conCtxt.lookup("myFileDataControl");
            Session session = rep.login();
            Property property;
            PropertyIterator contentPi;
            Document thisDoc;
            Node child;
            Node contentNode;
            try {
                Node node = (Node) session.getItem(sPath);
                NodeIterator iter = node.getNodes();
                PropertyIterator pi;
                while (iter.hasNext()) {
                    System.out.println("> fetching node");
                    child = iter.nextNode();
                    thisDoc = new Document();
                    thisDoc.setName(child.getName());
                    pi = child.getProperties();
                    System.out.println("*** "+child.getName());
                    while (pi.hasNext()) {
                        property = pi.nextProperty();
                        property =
                            child.getProperty("jcr:primaryType");
                        thisDoc.addProperty(property.getName(),
                            property.getValue().getString());
                    }
                    System.out.println("    Has Content:"
                        +child.hasNode("jcr:content"));
                    if (child.hasNode("jcr:content")) {
                        contentNode =
                            child.getNode("jcr:content");

```

```
        contentPi = contentNode.getProperties();
        while (contentPi.hasNext()) {
            property = contentPi.nextProperty();
            if (!property.getName().equals("jcr:data"))
                thisDoc.addProperty(property.getName(),
                    property.getValue().getString());
        }
        System.out.println("> adding doc to list");
        docs.add(thisDoc);
    }
    System.out.println("> done");
} finally {
    session.logout();
}
} catch (Exception e) {
    e.printStackTrace();
}
return docs;
}
}
```

Finally, you need to create a new data control out of the bean. Right-click the bean in Application Navigator and select Create Data Control.

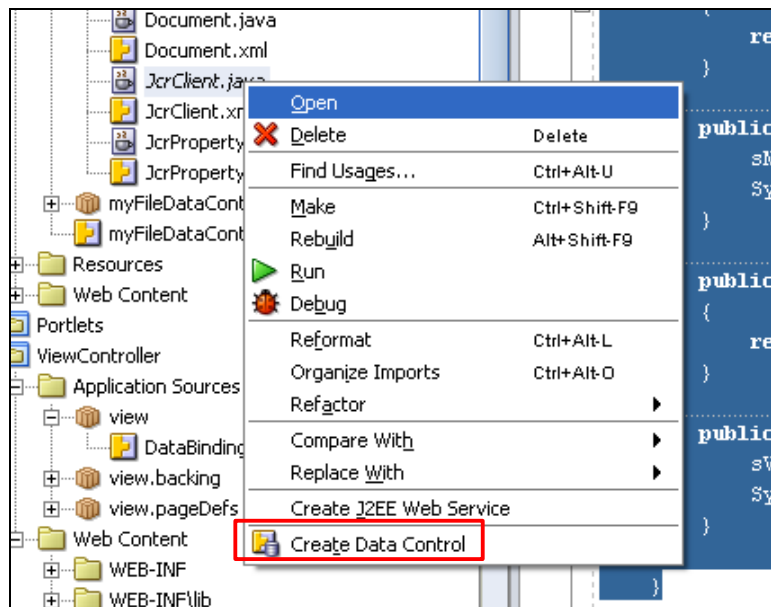


Figure 1. Select Create Data Control from the menu

Now you can create a new JSPX page and add selected elements from the data control to the page.

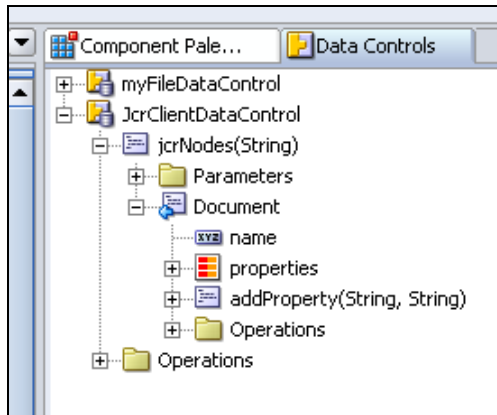


Figure 2. JcrClientDataControl in the Data Controls palette

First, drag and drop the Document on the page as an ADF table. Then take the “properties” element and drop it in the main cell of the ADF table as another ADF table.

Your page should look similar to this one:

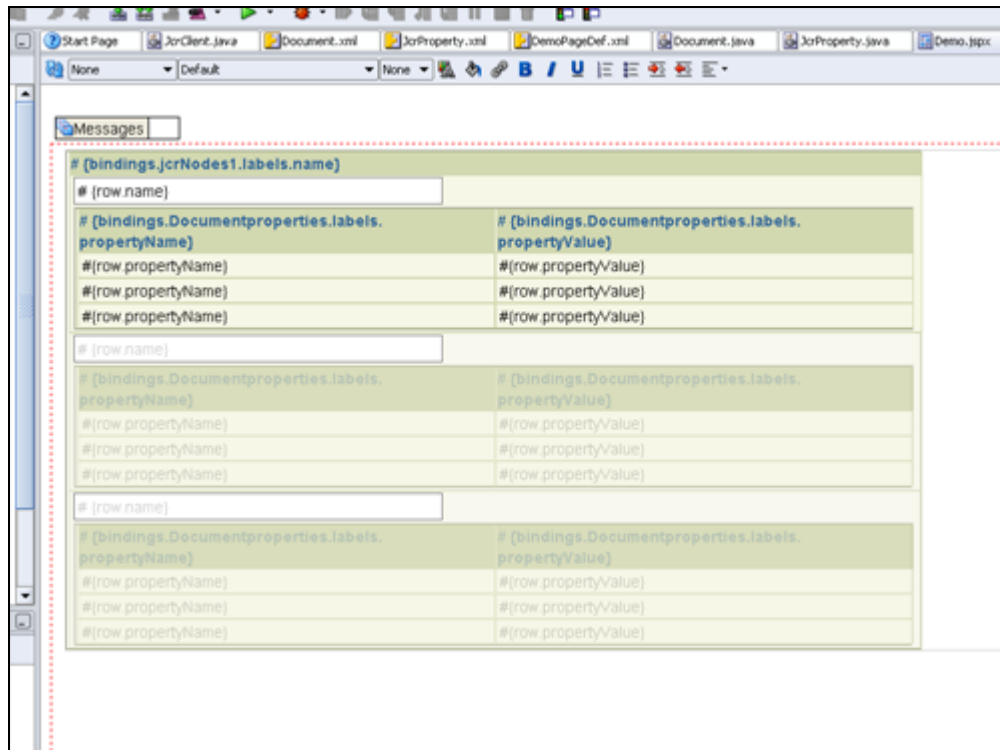


Figure 3. Design view of the page

Run the page. You should see all the files from the specified location and their properties, as shown in the Figure 4 below.

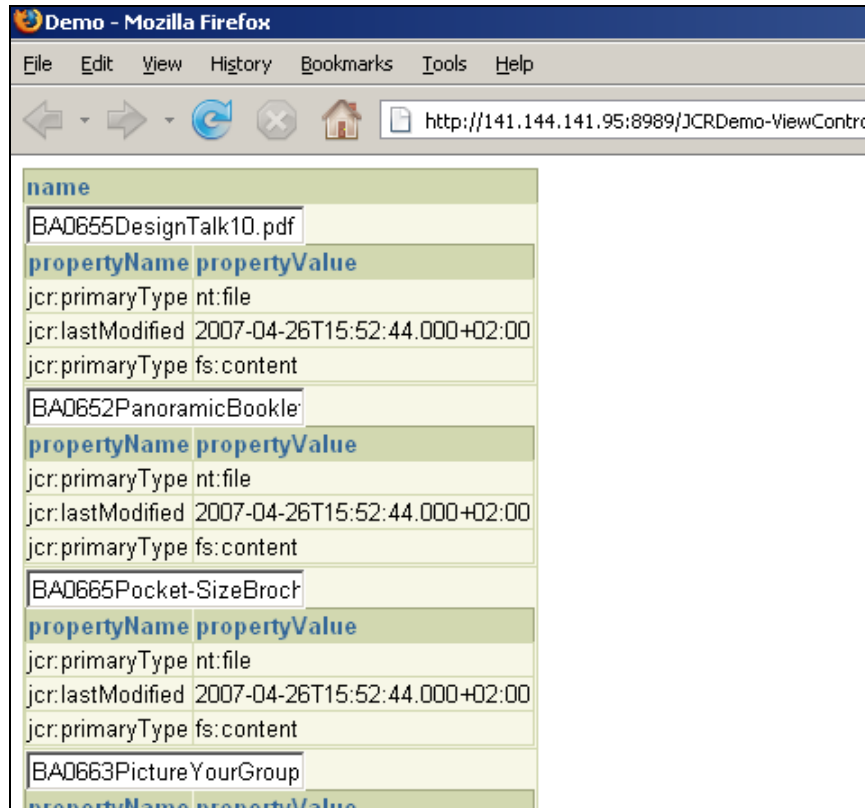


Figure 4. The running application

4. Summary

In your WebCenter application, you can create content repository data controls that connect to different repositories through adapters. A content repository data control provides access to the JCR APIs of the adapter instance it uses, so you can utilize the full power of JCR without having to know the connection and authorization details. This technical note showed how to use JCR to access content and its properties. Although our example accessed a file system, you can reuse the same code in your ADF/WebCenter applications to access a variety of content repositories.

ORACLE FUSION MIDDLEWARE

Oracle WebCenter TechNote

February 2008

Author: Philipp Weckerle

Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

oracle.com

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.