Developing WCM based WebSphere Portal application using IBM Rational Application Developer
Table of Content

Abstract ..........................................................................................................................3
Sample Use case ...........................................................................................................3
Prerequisite ..................................................................................................................3
Developing the portlet project ......................................................................................4
  Connecting to WCM repository in Source portlet .....................................................5
  Fetching the list of WCM libraries .............................................................................6
  Setting the default library and fetching the content ....................................................7
  Designing the view JSP for the Source portlet ..........................................................9
  Connecting to WCM repository in Target portlet ......................................................10
  Modifying the TargetPortletSessionbean ...............................................................10
  Designing the view JSP for the Target portlet ..........................................................11
  Enabling Source Portlet to publish the event ............................................................12
  Enabling Target Portlet to subscribe to the event ......................................................14
Deploying the portlet on WebSphere Portal .................................................................16
  Executing the use case on the server .......................................................................17
Summary ......................................................................................................................22
Resources ....................................................................................................................22
Abstract
Content management is playing an increasingly significant role in today's dynamic business world. The ease and efficiency to manage the web content throughout its lifecycle or workflows is the key to the success of a business sites. IBM® Web Content Manager included along with IBM WebSphere Portal provides a robust web content management solution to help customer manage their content from creation to rendering phase. You can extend the standard features of IBM® Web Content Manager (hereafter referred as WCM at some places) using the Web Content Manager API as well.

The aim of this article is to showcase development of Web Content Management (WCM) based portlets for WebSphere® Portal using IBM Rational® Application Developer V8.5. The article will cover how to use IBM Web Content Management API to build a portlet application and will also provide a sample application to help you using the IBM Web Content Management API that can be downloaded. This article assumes that you are aware of IBM Web Content Management and portlet development using IBM Rational Application Developer.

The sample use case of this article incorporates fetching the content from a content repository using WCM APIs.

Sample Use case
In this sample use case, you will work with a portlet project having two portlets that fetch and render the content from WCM libraries. These portlets would participate in inter-portlet communication by using the events mechanism provided by JSR 286 specification of portlet API.

The first portlet would list the content libraries available at server and based on the library selected it would fetch the content from the library and list the items in a HTML table. Content name in the first table would be a hyper-link, the second portlet would show the metadata and details of the content selected in first portlet. If the content selected in first portlet is an image resource type, it would be previewed in the second portlet, for file resource type content there would be a link provided in second portlet to save and open the resource along with its metadata.

Prerequisite
To create the sample, install the following software:
1) IBM Rational® Application Developer version 8.5
2) IBM WebSphere® Portal and Web Content Manager V8.0

On skill level, you should know how to work with IBM Rational Application Developer and portlet development to simplify the process of project creation.

You can prepare the development environment by installing IBM Rational Application Developer v8.5 and WebSphere Portal and Web Content Manager Server v8.0. You can import the sample project provided along with the article to IBM Rational® Application Developer workspace to have a detailed look. You need to import the content library to your Web Content Manager V8.0 server before you try to execute the sample on server. You can refer the detailed instructions at WCM infocenter.
Developing the portlet project

You would create a portlet project with two portlets. Use WCM_IPC_Sample as the portlet project name, Source as the first portlet name and Target as the names for the second portlet. To create the portlet project

Select File->New->Portlet Project. The Project creation dialog would launch

- Enter WCM_IPC_Sample as the name of the project name.
- Select WebSphere Portal v8.0 as your target runtime.
- Select the Create a portlet checkbox.
- Enter the Portlet name as Source.
- Click the modify button of the Configuration label, and select JSR286 as portlet API and Basic portlet as Portlet type.
- Click Finish.

Figure 1: New portlet project wizard
The portlet project containing the Source portlet would be created in the workspace. To create the second portlet i.e. Target portlet, Select File->New->Portlet. The Portlet creation dialog would launch
- Enter the Portlet name as Source.
- Select Finish

Figure 2: New portlet wizard

The portlet project WCM_IPC_Sample would now contain two portlets, Source portlet and Target portlet.

Connecting to WCM repository in Source portlet

Workspace is the core of the IBM® Web Content Manager API. Content is created, saved, deleted and searched for in scope of a workspace item. A workspace is associated to a user and is an interface to Web Content Manager. You would request for a workspace object from repository singleton.

You would create the connection to the WCM repository in the init phase of the portlet itself. The init method of the portlet class would look like the code snippet below:

Code Snippet 1: Init method of Source portlet
public void init() throws PortletException {
    super.init();
    try {
        myworkspace = WCM_API.getRepository().getWorkspace();
        if (myworkspace != null) {
            myworkspace.login();
        } else {
            System.out.println("Unable to get a valid workspace.<br/>");
        }
    } catch (Exception e) {
        System.out.println("Exception " + e.getMessage());
        e.printStackTrace();
    }
}

The statement WCM_API.getRepository().getWorkspace() is responsible for fetching a workspace object to interface with WCM.

Fetching the list of WCM libraries

There can be multiple document libraries available at server as per the configuration. To let the user choose the desired library you need to fetch the list of available libraries from WCM workspace and store the list of library names. To store the list you can define an ArrayList ‘libraries’ as a list of strings and create the desired getter and setters for the same. As the list of libraries is to be persisted at session level you would opt to use the session bean SourcePortletSessionBean generated along with portlet project by RAD. At a later stage while developing the GUI for the Source portlet you would map this list of document libraries to the dropdown to let user select the desired library.

In portlet session bean SourcePortletSessionBean you would define an ArrayList to store the list of libraries, along with its getters and setters.

Code Snippet 2: ArrayList to store the list of libraries in SourcePortletSessionBean

private List<String> libraries = new ArrayList();

public void setLibraries(String libText) {
    libraries.add(libText);
}

class getLibraries() {
    return this.libraries;
}

To get the list of available document libraries in WCM, you would add the desired piece of code to the getSessionBean method of the source portlet class.

Code Snippet 3: getSessionBean method to fetch the list of libraries in SourcePortlet.java

//Populate the drop down with list of libraries in workspace
Iterator<DocumentLibrary> myLibraries = myworkspace.getDocumentLibraries();
if (myLibraries.hasNext()) {
    while (myLibraries.hasNext()) {
        sessionBean.setLibraries(myLibraries.next().getName());
    }
}
myworkspace.getDocumentLibraries() would fetch the list of libraries available in the workspace and then sessionBean.setLibraries is storing the library names to the list iteratively.

**Setting the default library and fetching the content**

To allow selection of document library and fetch the contents from the selected library, you can refer the following snippet in processAction method in SourcePortlet class.

**Code Snippet 4: Snippet from processAction method in SourcePortlet.java**

```java
try {
    DocumentLibrary myLibrary =
    myworkspace.getDocumentLibrary(request.getParameter(FORM_DROPDOWN).toString());
    if (myLibrary != null) {
        sessionBean.populatedocList(myworkspace, myLibrary);
    } // end of MyLibrary if
    else
        System.out.println("Library is null<br/>");
} //end of try block
catch (Exception e) {
    // Normally exceptions would be added to logs
    System.out.println("Exception " + e.getMessage());
    e.printStackTrace();
} //end of catch block
```

The method populatedocList is implemented in SourcePortletSessionBean and is intended to set the selected library as the workspace library and then search the content from the selected library. You can refer the following snippet in populatedocList method of SourcePortletSessionBean.

**Code Snippet 5: Snippet from populatedocList method of SourcePortletSessionBean**

```java
myworkspace.setCurrentDocumentLibrary(myLibrary);
QueryService queryService = myworkspace.getQueryService();
Query query = queryService.createQuery();
query.addSelector(Selectors.libraryEquals(myworkspace.getCurrentDocumentLibrary()));
query.addSelector(Selectors.typeIn(DocumentTypes.LibraryFileComponent.getApiType(),DocumentTypes.LibraryImageComponent.getApiType()));
ResultIterator resultIterator = queryService.execute(query);
```

The statement myworkspace.setCurrentDocumentLibrary(myLibrary) would set the selected library to the current workspace library. Then you would use QueryService to retrieve WCM Document by DocumentQuery. For the use case we are limiting the scope to only two DocumentTypes i.e. `LibraryFileComponent` and `LibraryImageComponent`. The result set of query execution would be stored in an iterator of WCM API query results i.e. resultIterator.

The code snippet below would parse the result set and get the desired properties for the nodes returned in result which would be referenced to render the details. You would store these details in a node docList and then final collection of these nodes would be maintained in another ArrayList of these nodes.

**Code Snippet 6: Snippet from populatedocList method of SourcePortletSessionBean**
if (resultIterator.hasNext()) {
    if (resultIterator.hasNext()) {
        WCMApiObject resource = (WCMApiObject) resultIterator.next();
        DocumentList docList = new DocumentList();
        resourceName = resource.getName();
        if (resource instanceof LibraryImageComponent) {
            resourceURL = ((LibraryImageComponent) resource)
                .getResourceURL();
            resourceCreator = ((LibraryImageComponent) resource)
                .getCreator();
            resourceDate = ((LibraryImageComponent) resource)
                .getCreationDate().toString();
            docList.setDocumentList(resourceName, resourceURL, resourceCreator,
                resourceDate, LIB_IMAGE_COMP);
            setResult(docList);
        } else if (resource instanceof LibraryFileComponent) {
            resourceURL = ((LibraryFileComponent) resource)
                .getResourceURL();
            resourceCreator = ((LibraryFileComponent) resource)
                .getCreator();
            resourceDate = ((LibraryFileComponent) resource)
                .getCreationDate().toString();
            docList.setDocumentList(resourceName, resourceURL, resourceCreator,
                resourceDate, LIB_FILE_COMP);
            setResult(docList);
        } else {  
            System.out.println("No objects Found");
        }
    } //end of while
} //end of resultIterator if
else {

}  //end of resultIterator else

The SourcePortletSessionBean has another ArrayList result defined to store the result with additional metadata that is required to display the details.

Code Snippet 7: Snippet from SourcePortletSessionBean

```java
public List<DocumentList> result = new ArrayList();

public void setResult(DocumentList obj) {
    result.add(obj);
}

public List getResult() {
    return this.result;
}
```

The element result is an ArrayList of type DocumentList. DocumentList is an additional bean defined to store the desired metadata of the content items. The snippet above references an element type DocumentList, setDocumentList is the setter method to save the metadata for this additional bean.

Code Snippet 8: Snippet from DocumentList.java

```java
public void setDocumentList(String resName, String resURL, String resAuthors,
    String resDate, String resType) {
```
The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

```java
this.name = resName;
this.url = resURL;
this.authors = resAuthors;
this.date = resDate;
this.type = resType;
}

public DocumentList getDocumentList() {
    return (DocumentList) this;
}

Designing the view JSP for the Source portlet

The objective of source portlet is to let user make a selection of WCM library and based on library selected, fetch and display the list of contents in a HTML table. You can refer the SourcePortletView.jsp in attached sample file to get the JSP details.

**Code Snippet 9: Snippet from SourcePortletView.jsp to display list of libraries**

```jsp
<% String formText = sessionBean.getFormText();
    Iterator libLst = sessionBean.getLibraries().iterator();
    Iterator resLst = sessionBean.getResult().iterator(); %>

<FORM method="POST" action="<portlet:actionURL/>">
   <LABEL for="<%=com.ibm.wcm_ipc_sample.SourcePortlet.FORM_TEXT%>" Select the WCM library :</LABEL><BR>
   <SELECT name="<%=com.ibm.wcm_ipc_sample.SourcePortlet.FORM_DROPDOWN%>">
      <option value="">Select a Library</option>
      <% while (libLst.hasNext()) {
         String library = (String)libLst.next();%>
      <option value="<%= library %>"><%= library %></option>
      <% } %>
   </SELECT>
   <INPUT name="<%=com.ibm.wcm_ipc_sample.SourcePortlet.FORM_SUBMIT%>" type="submit" value="Submit"/>
</FORM>
</DIV>
```

The code snippet above provides a dropdown with the list of libraries values populated using sessionBean.getLibraries().iterator(). Once you make a selection and click Submit, the value of library would be passed to populateList method through processAction method of SourcePortlet class to set it as a current library of workspace.

To display the list of result set generated post selecting the library, you can refer the following snippet.

**Code Snippet 10: Snippet from SourcePortletView.jsp to display the content**

```jsp
<% if (resLst.hasNext()) { %>
   <table border="1" cellpadding="3" cellspacing="2">
      <tr bgcolor="#c0c0c0">
         <th align="center" valign="middle">Name</th>
         <th align="center" valign="middle">Authors</th>
         <th align="center" valign="middle">Creation Date</th>
      </tr>
      <tr bgcolor="#c0c0c0">
```

The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.
The result iterator would loop to place the nodes in list in a HTML table format. The first column would display the name and would be a hyperlink, its value would be a concatenated string of its metadata separated by :: “double hash”. This string would be passed on to target portlet to display the properties of a selected item to depict the inter-portlet communication.

**Connecting to WCM repository in Target portlet**

Similar to Source portlet you would create the connection to the WCM repository in the init method of the Target portlet itself to interface with the WCM.

**Modifying the TargetPortletSessionbean**

TargetPortletSessionbean has a method setDisplayObject to contain the logic to parse the concatenated string that is passed by the source JSP (as a value of event to achieve inter-portlet communication). It would decipher the string to display the selected content’s metadata. You can refer the source in the attached sample.

---

The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.
The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Code Snippet 11: Snippet from TargetPortletSessionBean

```java
public void setDisplayObject(String formObj) {
    displayObj.setName(formObj.substring(0, formObj.indexOf("::")));
    formObj = formObj.replace(displayObj.getName()+"::", "");
    displayObj.setAuthors(formObj.substring(0, formObj.indexOf("::")));
    formObj = formObj.replace(displayObj.getAuthors()+"::", "");
    displayObj.setDate(formObj.substring(0, formObj.indexOf("::")));
    formObj = formObj.replace(displayObj.getDate()+"::", "");
    displayObj.setUrl(formObj.substring(0, formObj.indexOf("::")));
    formObj = formObj.replace(displayObj.getUrl()+"::", "");
    displayObj.setType(formObj);
}
```

Designing the view JSP for the Target portlet

The view JSP for target portlet would fetch the details of the object to display using the call `sessionBean.getDisplayObject()`. For image resource types it would preview the image along with its metadata in a HTML table format and for file resources it would render the hyperlink for the resource along with its metadata in a HTML table format.

Code Snippet 12: Snippet from TargetPortletView.jsp

```jsp
<%@
DocumentList disp = sessionBean.getDisplayObject();
if (!disp.getName().isEmpty()) {
    if (disp.getType().matches("LibraryImageComponent")) {
        <table border="1" cellspacing="1" cellpadding="2">
            <tr bgcolor="#F5F5F5">
                <td colspan="2" align="center"><img src="<%=disp.getUrl()%>" width="510" height="300"></td>
            </tr>
            <tr bgcolor="#c0c0c0">
                <th align="center" valign="middle" width="250">Document Name</th>
                <td width="250"><%=disp.getName()%></td>
            </tr>
            <tr bgcolor="#F5F5F5">
                <th align="center" valign="middle" width="250">Document Creators</th>
                <td width="250"><%=disp.getAuthors()%></td>
            </tr>
            <tr bgcolor="#c0c0c0">
                <th align="center" valign="middle" width="250">Document Type</th>
                <td width="250"><%=disp.getType()%></td>
            </tr>
        </table>
    } else {
        <table border="1" cellspacing="3" cellpadding="2">
            <tr bgcolor="#F5F5F5">
                <th align="center" valign="middle" width="250">Document Details</th>
            </tr>
            <tr bgcolor="#c0c0c0">
                <th align="center" valign="middle" width="250">Document Name & Link</th>
            </tr>
        </table>
    }
}</%@>
Enabling Source Portlet to publish the event

Finally you would setup the events mechanism to let the two portlets i.e. Source portlet and Target portlet communicate to each other. To set the events mechanism in Source portlet,

- Open the view JSP for Source portlet
- Drag and drop the Event Source Trigger from the portlet drawer in palette view on to the Name column in HTML table.

Figure 3: Event Source Trigger option in palette view

- Insert Event Source Trigger wizard will open.
Figure 4: Insert event source trigger wizard

- Select the source portlet as the Source.
- Click the New Event button against the Event name.
- Describe the Event being published dialog will open
The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Figure 5: Enable this portlet to publish event wizard

- Input the event name as documentName.
- Select Finish keeping rest of the values as default.

It would modify the processAction method in SourcePortlet class with the following snippet.

Code Snippet 13: Snippet from processAction method in SourcePortlet class

```java
if (request.getParameter(DOCUMENTNAME_PARAM) != null) {
    response.setEvent("documentName", request.getParameter(DOCUMENTNAME_PARAM));
}
```

Enabling Target Portlet to subscribe to the event

Now you would enable the target portlet to process the event send by the Source portlet. To enable the target portlet to subscribe to the event published by source portlet.

- Expand the Portlet Deployment Descriptor node in Enterprise Explorer view of the workspace. Select the target portlet.
- In the context menu, select Events -> Enable this portlet to Process Event.
- Describe the Event being Processed dialog would open.
The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Figure 6: Enable this portlet to process the event wizard

- Select the same event name from the drop down, which was used in source portlet i.e. documentName
- Press Finish.

It would add a method processEvent in portlet class for target portlet. You need to modify that to ensure it meets the objective for your usecase. Refer the code snippet below.

**Code Snippet 14: Snippet from processEvent method in TargetPortlet class**

```java
Event sampleEvent = request.getEvent();
if(sampleEvent.getName().toString().equals("documentName")) {
    Object DocumentDetails = sampleEvent.getValue();
    TargetPortletSessionBean sessionBean = getSessionBean(request);
    if( sessionBean != null )
        sessionBean.setDisplayObject(DocumentDetails.toString());
}
```

In the snippet above, on receiving the documentName event, it would fetch the value received and invoke the method sessionBean.setDisplayObject to parse the value passed by the source portlet.
Deploying the portlet on WebSphere Portal

Now you are all set to publish the portlet application on to WebSphere Portal. To publish the project, create the server instance and start the server, in case it is not started. To create the server instance:

- Go to File -> New -> Other -> Server to launch the server creation wizard
- Select WebSphere Portal v8.0 Server and provide the hostname, click Next.

**Figure 7: Server creation wizard**

- On the WebSphere Settings wizard, provide the port numbers to connect to the server, server credentials, and click Next.
- On the Portal Settings wizard, provide the server credentials required to publish the project to the server, and click Next.
- Click Finish.
- If server is not started, right-click the server instance created above and select Start.
- After the server starts, right-click on portlet project and select Run on server to display the Run
On Server wizard.

- In the Run On Server wizard, select the server instance that you created and click Finish.

Figure 8: Run On Server option

Executing the use case on the server

The Run On Server wizard publishes the portlet project. It also places the portlet on a portal page labeled as Rational Components, and opens the browser instance rendering the portlet. You will have the portlets Source and Target on the page.
The sample program is provided to you on an "AS IS" basis, without warranty of any kind.
IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Figure 9: Output displaying the source and target portlets

Source portlet would have a dropdown showing the list of available WCM libraries in the server.

Figure 10: Output with Select library dropdown expanded

Select a library and press Submit button to display the content list matching the filter criteria set in populate method.
Before you can witness inter-portlet communication, you need to wire the source and target portlets on server. To wire the two portlets (refer figure 12)

- Select Edit Page Layout for the page on which the portlets are placed. To traverse to the page go to Administration > Portal User Interface > Manage Pages > Content Root > Home and select the page where portlet is placed.
- Select the Wires tab.
- For the source portlet, select Source from the drop-down menu, and for the target portlet, select the Target portlet. For the event names, click the drop-down arrow for the Sending and Receiving fields.
- Click the plus sign and then click Done.
Figure 12: Wire the source and target portlets

Once you have wired the portlets, select a content Name in source portlet (for example, the first row in the Source Portlet). When you click the Name, an event displayName is triggered by the Source portlet, which sends the value (i.e. concatenated metadata) to Target portlet. On receiving the event, the Target Portlet processes the event. It fetches the value and parses and displays the same for the user in a HTML table.
The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.
Summary

In this article you have learned development of Web Content Management (WCM) based portlets for WebSphere® Portal by using IBM Rational Application Developer. The use case that you followed created a portlet project in IBM Rational Application Developer, and showed how to use WCM API to interface with WCM using a workspace object. You fetched the list of content libraries before setting one of the library as a current workspace library. Then you queried for the content in a library based on the document type. This was followed by learning how easily eventing can be achieved between two portlets using IBM Rational Application Developer, where Source portlet was configured to publish an event and Target portlet was configured to subscribe and process the same event. Finally, to visualize and test the application, you published the portlet project to WebSphere Portal Server and wire the two portlets using the wiring tool provided by WebSphere Portal Server.

Resources

1) Get information about Rational® Application Developer and how to use it:
   - IBM Rational Application Developer v85 Information center
   - IBM Rational application Developer wiki – Portal tools
2) For product documentation about IBM Web Content Manager v 8.0
   - IBM Web Content Manager 8 Product Documentation
   - Troubleshooting WCM API, How To, and Sample Code