

Portal User Management Architecture (PUMA) sample scenarios in IBM® WebSphere® Portal

[James W Barnes](#), *WebSphere Portal API Level 2 Team Lead*

[Thomas Hurek](#), *Chief Programmer, Fix Packs & Lab Services Consultant*

[Ryan R Wilson](#), *WebSphere Portal API Level 2 Technical Lead*

*WebSphere Portal Development and Support
IBM US & Germany*

September 2008

© Copyright International Business Machines Corporation 2008. All rights reserved.

Abstract. This document describes how to use Portal User Management Architecture (PUMA) for your IBM® WebSphere® Portal implementation. Specifically, you learn how to use a public API to implement custom scenarios with code samples, and how to customize the existing forms and screens for custom user management. It is intended for WebSphere Portal application developers and administrators who need to implement custom solutions to suit the needs for individual user management.

Contents

1	Introduction to WebSphere Portal user management	2
1.1	Portal User Management Architecture.....	3
1.2	Supported user registries.....	3
1.3	User management operations.....	4
1.4	PUMA configuration	4
1.5	WMM configuration	5
1.6	PUMA programming interface	5
2	Implementation scenarios for PUMA	6
2.1	Creating a user management portlet using the PUMA SPI.....	6
2.2	Adding a user	10
2.3	Editing a user	14
2.4	Deleting a user	15
2.5	Adding a group	16
2.6	Deleting a group.....	17
2.7	Adding a member to a group	18
2.8	Removing a member from a group	21
2.9	Customizing password expiration.....	21
2.9.1	Alternate method for retrieving the data.....	31
3	Implementation scenario: User Agreement use case	32
3.1	User Agreement form	32
3.2	Custom Log-in portlet.....	33
3.3	Determining whether a user has signed the agreement.....	36
3.3.1	SingletonUserAgreement.....	36
3.4	Redirecting the user to the Agreement page	38
3.5	Displaying the User Agreement form in the log-in portlet	39
4	Conclusion	41
5	Resources	41
6	About the authors.....	41

1 Introduction to WebSphere Portal user management

WebSphere Portal features a powerful user management component that lets you connect it to various user repositories. Although these user repositories contain the actual users and groups data, it is the Portal User Management Architecture (PUMA) component that is key to manipulating the data in the user repositories for the users of WebSphere Portal.

We begin with an introduction to PUMA and the operations and configurations that are possible. The main part of the paper describes how to use a public API to implement custom scenarios with code samples, and how to customize the existing forms and screens for custom user management.

Among the forms and screens is the User Agreement form, the Password expiration screen, and the Self Registration/Edit Profile screens. Finally, we create a User Agreement form use case that forces users to sign before accessing the protected pages.

To get the most from this white paper, you should have a general understanding of WebSphere Portal administration and security setup. Although this paper focuses on version 6.0.x, the main

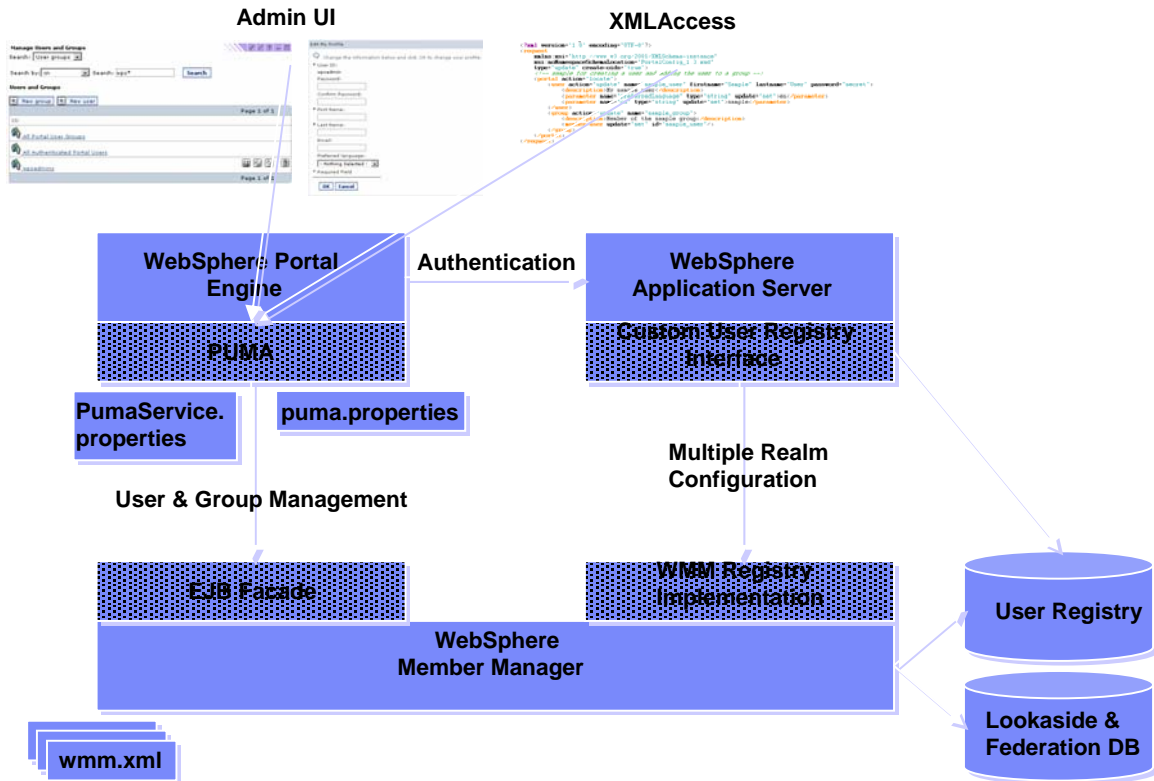
PUMA scenarios

concepts also apply to versions 5.1.0.x and 6.1.0.x. More information about WebSphere Portal administration and the user management basics can be found in the [WebSphere Portal Information Center](#).

1.1 Portal User Management Architecture

Figure 1 shows how the PUMA component fits within the WebSphere Portal architecture.

Figure 1. WebSphere Portal architecture showing PUMA



User management in WebSphere Portal can be performed either through the user interface Administration portlets (for example, via the User and Groups portlet) or via the scripting interface XMLAccess that calls the user management component for the operations.

The user management component facilitates the WebSphere Member Manager (WMM) component as an abstraction layer to the physical user registry. WMM connects to the User Registry and optionally to a federation and lookaside database. See the next section for details about the supported user registries.

The configuration of the PUMA and WMM components for the different user repositories is done via property or XML files. If multiple realms are configured, WMM is also used for the authentication of users.

1.2 Supported user registries

WebSphere Portal supports the use of the WMM user database, a Lightweight Directory Access Protocol (LDAP) server, or a custom user registry (CUR) as user registries. The WMM user database stores the user and group information within the configured database system. See the

[WebSphere Portal detailed system requirements](#) page for details on which LDAP servers are supported.

A CUR is a custom implementation that might use a database or other backend systems to store the data. After WebSphere is installed, the WMM user database is used as user registry, and security is disabled in the WebSphere Application Server configuration. This configuration is not recommended for production use due to security risks.

The configuration tasks `enable-security-ldap` or `enable-security-wmmur-ldap` allow you to enable security and configuring the specified LDAP server.

The configuration task `enable-security-wmmur-db` enables security and configures the WMM user database as the user registry.

1.3 User management operations

WebSphere Portal lets you perform the following user management operations:

- Users:
 - Creation
 - Self-Enrollment: Anonymous user can create a user
 - Modification of attributes (among these: password)
 - Selfcare: User can update his own attributes
 - Deletion
- Groups:
 - Creation
 - Modification of attributes
 - Deletion
 - Adding a user to a group
 - Removing a user from a group

All these operations are protected with WebSphere Portal Access Control to ensure that only privileged users can perform the operations.

The following interfaces allow the different operations:

- XMLAccess: Does not allow Self-Enrollment.
- Administration portlets: User and Groups Portlet, Edit my Profile Portlet and Screen
- PUMA programming interface: Allows writing custom implementations that trigger user management operations via the programming interface.

1.4 PUMA configuration

The configuration of the PUMA and the WMM components determines the effect of the user management operations in the user repository. While the WMM component configuration is responsible for the connection to the user repository, the PUMA component configuration determines the way users and groups are retrieved from and stored in the user repository.

The configuration of the PUMA component is done via the `PumaService.properties` (located in `PortalServer/shared/app/config/services/`) and `puma.properties` (located in `PortalServer/shared/app/config/`) files. In Portal 6.0.x the configuration of the properties is handled via the WebSphere Portal PumaService in the WAS Admin Console.

1.5 WMM configuration

You must configure WMM by editing the `wmm.xml`, `wmmAttributes.xml`, `wmmLDAPServerAttributes.xml`, `wmmDBAttributes.xml`, `wmmur.xml`, and `wmmWASAdmin.xml` files, all of which are stored in the Portal/wmm directory. In a cluster environment you must be sure to first check out the files from the Deployment Manager (using the `check-out-wmm-cfg-files-from-dmgr` config task), and after making the change in the Portal/wmm directory, to check them in again (using the `check-in-wmm-cfg-files-to-dmgr` config task).

Here's what the files do:

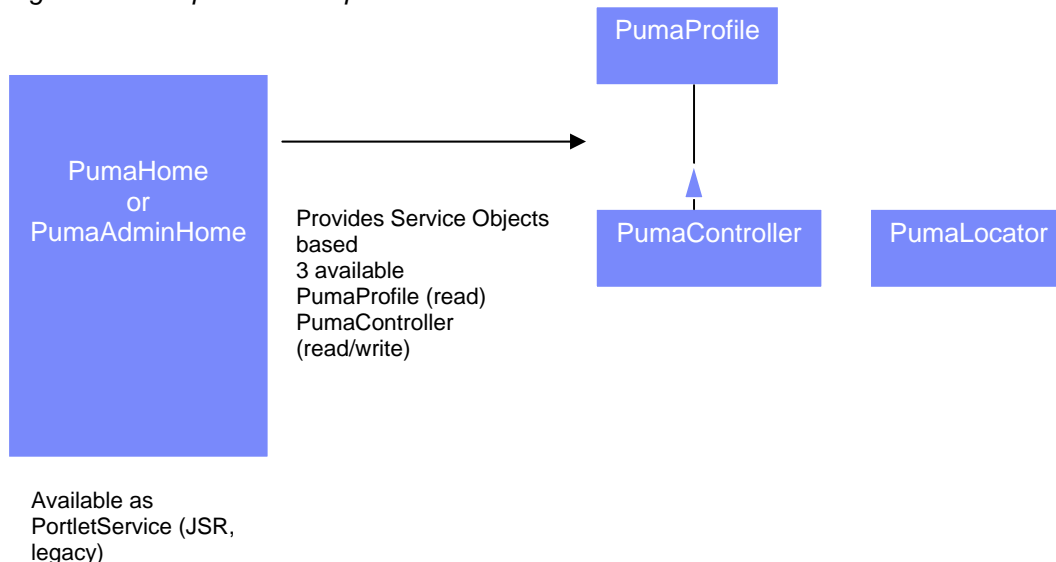
- `wmm.xml`: Stores information about the repositories.
- `wmmAttributes.xml`: Basic definition of the attributes supported by WMM.
- `wmmLDAPServerAttributes.xml`: Default mapping of attributes for the LDAP server.
- `wmmDBAttributes.xml`: Default mapping of attributes when using the database repository.
- `wmmUR.xml`: Contains the "realm" information.
- `wmmWASAdmin.xml`: Defines the user base in the File local mode case (used during startup of the server).

1.6 PUMA programming interface

WebSphere Portal Version 5.1.0.1 introduced a System Programming Interface (SPI) for managing user profile and group membership information, called PUMA SPI. You can use it to create, update, and delete users and groups, and to retrieve information about the current active user. It also lets you search for users and groups in your backend repository.

The PUMA SPI consists of a Home object, available through JNDI lookup, and three service interfaces, available through the Home object (see figure 2).

Figure 2. WebSphere Portal public PUMA SPI



While the PUMA SPI takes care of handling Java™ 2 Security permissions under the cover, it is protected via WebSphere Portal Access Control. This means that the user who is accessing the

portlet or theme that uses the PUMA SPI must have the appropriate permissions. If this access is on an anonymous page, the anonymous user would need to be given these access rights. As a result, we introduced the PumaAdminHome in WebSphere Portal 6.0; when this interface is used, no permissions are required for the executing user.

The PumaHome classes are available via different packages to be accessed from a JSR Portlet (package com.ibm.portal.um.portletservice.*), Legacy Portlet (package com.ibm.portal.um.portletservice.legacy.*), and in other code areas such as in a Theme or Skin or EJB (package com.ibm.portal.um.*). The PumaAdminHome is available only in the package com.ibm.portal.um.* but can also be accessed via a portlet.

In the programming, you will first get the PumaHome or PumaAdminHome via JNDI lookup, then from it get the provider object, and then use it to perform the according operations. The samples in the following sections are designed help you learn how to use the PUMA SPI.

2 Implementation scenarios for PUMA

This section describes how to use PUMA for your WebSphere Portal implementation, focusing on implementation coding examples using the PUMA SPI. The two examples are a user management portlet and a password expiration sample.

2.1 Creating a user management portlet using the PUMA SPI

First, you must decide what you intend to do via your user management portlet. In this example, we wanted to be able to create users, create groups, delete users, delete groups, add members to groups, delete members from groups, and edit users.

For performance, we first set up a class variable for the PumaServiceHome:

```
private PortletServiceHome psh;
```

Then in the init method we perform this lookup once so that we save on performance later:

```
try {
    if(psh == null) {
        javax.naming.Context ctx = new javax.naming.InitialContext();
        psh = (PortletServiceHome)
            ctx.lookup("portletservice/com.ibm.portal.um.portletservice.PumaHome");
    }
} catch (NamingException e) {
    System.err.println("The service home could not be retrieved = "
        + e);
}
```

Finally we set up two class variables to hold the default set of attributes for users and groups. These will be used to do the basic search for users and groups.

```
private ArrayList USER_ATTRS = null;
private ArrayList GROUP_ATTRS = null;
```

We initialize them in the init method as well:

```
USER_ATTRS = new ArrayList();
```

PUMA scenarios

```
USER_ATTRS.add("givenName");
USER_ATTRS.add("sn");
USER_ATTRS.add("uid");
GROUP_ATTRS = new ArrayList();
GROUP_ATTRS.add("cn");
```

This remaining section of code is in the doview method of the portlet. The first thing that we must do as part of the user management portlet is to search for users and groups, but first we must retrieve the PumaHome from the PumaServiceHome:

```
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
```

With that we also must retrieve the PumaProfile and the PumaLocator:

```
PumaProfile pp = service.getProfile(request);
PumaLocator pl = service.getLocator(request);
```

Once we have these objects we can start searching for users or groups. First we look for groups, which returns an array list of the CN's for the groups. To find these groups, we use the PumaLocator to search for groups by attribute.

Then, as we get back a list of group objects, we need to iterate over that list and use the PumaProfile to get a more meaningful display attribute. Some of the attributes we receive can be multi-valued, so we must see if it is an actual string or if it is a array of objects first:

```
ArrayList newGroupList = new ArrayList();
ArrayList attrArray = null;
String groupName = "";
Map groupMap = null;
List userGroupsList =
pl.findGroupsByAttribute(request.getParameter("searchBy"),
request.getParameter("searchString"+"*");
for (int i = 0; i<userGroupsList.size(); i++) {
groupMap = pp.getAttributes((Group)userGroupsList.get(i), GROUP_ATTRS);
Object attrObj = groupMap.get("cn");
if (attrObj instanceof java.util.List) {
attrArray = (ArrayList)attrObj;
groupName = (String)attrArray.get(0);
} else {
groupName = (String) attrObj;
}
newGroupList.add(groupName);
}
request.setAttribute("groupList", newGroupList);
```

Next we add in the search for users. Just as before, it searches for the attribute request with the search string that was requested. Next, just as with the group object that we retrieved previously, the user object that is returned has no methods to display any attributes, so we iterate through the list, using the PumaProfile to get attributes for that, and return an Arraylist of the UserViewBean:

PUMA scenarios

```
List userList =
pl.findUsersByAttribute(request.getParameter("searchBy"),
request.getParameter("searchString"+"*");
request.setAttribute("userList", retrieveUserAttributes(userList, pp));
```

```
private ArrayList retrieveUserAttributes(List userList, PumaProfile
pp)
throws PumaModelException,
PumaSystemException,
PumaAttributeException,
PumaMissingAccessRightsException {
    UserViewBean newUser = null;
    Map userMap = null;
    ArrayList newUserList = new ArrayList();
    ArrayList attrArray = null;
    for (int i = 0; i<userList.size(); i++) {
        newUser = new UserViewBean();
        userMap = pp.getAttributes((User)userList.get(i), USER_ATTRS);
        Object attrObj = userMap.get("givenName");
        if (attrObj instanceof java.util.List) {
            attrArray = (ArrayList)attrObj;
            newUser.setGivenName((String)attrArray.get(0));
        } else {
            newUser.setGivenName((String) attrObj);
        }
        attrObj = userMap.get("sn");
        if (attrObj instanceof java.util.List) {
            attrArray = (ArrayList)attrObj;
            newUser.setSn((String)attrArray.get(0));
        } else {
            newUser.setSn((String) attrObj);
        }
        attrObj = userMap.get("uid");
        if (attrObj instanceof java.util.List) {
            attrArray = (ArrayList)attrObj;
            newUser.setUid((String)attrArray.get(0));
        } else {
            newUser.setUid((String) attrObj);
        }
        newUserList.add(newUser);
    }
    return newUserList;
}
```

Now that we can retrieve users and groups, we need to add a form to the .jsp so that we can enter the search criteria. The searchBy is populated from a JavaScript™ that is included in the .jsp, so refer to the script block in the UserManagementPortletView.jsp:

```
<div id="searchBox">
<portlet:actionURL var="searchUrl">
<portlet:param name="action" value="searchLdap" />
</portlet:actionURL>

<form action="<%=searchUrl %>" name="testform" id="testform">
```


PUMA scenarios

```
Search: <select name="<portlet:namespace/>searchType"
onchange="populate()">
<option value="userGroups">User Groups</option>
<option value="users">Users</option>
</select><br>
Search by: <select name="<portlet:namespace/>searchBy">
<option value="cn">CN</option>
<option value="description">description</option>
</select><br>

Search String: <input type="text" name="searchString" size="20">

<input type="submit" name="Search" value="Search">
</form>

</div>
```

Now we must add the display area to show the users and groups that are returned from the code that we added previously:

```
<div id="searchResults">
<%List userGroupsList = (List)renderRequest.getAttribute("groupList");
List userList = (List)renderRequest.getAttribute("userList");
if(userGroupsList != null) {
%><table border=1> <%
String myGroup = null;
for (int i = 0; i < userGroupsList.size(); i++) {
myGroup = (String) userGroupsList.get(i); %>
<tr>
<td width="80%"><img border="0"
src='<%=renderResponse.encodeURL(renderRequest.getContextPath()+"/image
s/group_icon.gif") %>' />
<%=myGroup %></td>
</tr>
<% }%>
</table> <%
}%>
<p>
<% if(userList != null) { %>
<table border=1> <%
UserViewBean myUser = null;
for (int i = 0; i < userList.size(); i++) {
myUser = (UserViewBean) userList.get(i); %>
<tr>
<td width="75%">
<%=myUser.getGivenName() %> <%=myUser.getSn() %></td>
</tr>
<% }%>
</table> <%
}%>
</div>
```

Because the form is posting to an action URL, we need to add some components to the processAction to ensure these parameters are passed along:

PUMA scenarios

```
if( action != null ) {
if(action.equals("searchLdap")) {
response.setRenderParameter("searchType",
request.getParameter(namespace+"searchType"));
response.setRenderParameter("searchBy",
request.getParameter(namespace+"searchBy"));
response.setRenderParameter("searchString",
request.getParameter("searchString"));
}
```

The code discussed thus far forms the basis for the user management portlet. With it we can now search for users and groups and display those results onto the screen. Next we look at the individual functional pieces you may wish to add for create, read, update, and delete (CRUD) actions on users and groups.

2.2 Adding a user

First we focus on the code needed to add a user. To the initial view .jsp we add a button to take us to the Add User page:

```
<portlet:renderURL var="newUser">
<portlet:param name="navPage" value="newUser" />
</portlet:renderURL>
<a nowrap="" href="<%=newUser %>" style="text-decoration: none;">
<img class="wpsDialogIcon" border="0" alt="New User" title="New User"
src='<%=renderResponse.encodeURL(renderRequest.getContextPath()+"/images/
New_Task.gif") %>' />
</a>
```

That link tells the portlet to navigate to the Add User page. To make this form more dynamic, we use the LanguageProvider from the model SPI to get a list of languages that the portal supports.

If you don't want to do it this way, an OldAddUser.jsp is provided that has only a list of languages. If you want to make this list of attributes more dynamic, you could use the method `getDefinedUserAttributeNames()` on the `PumaProfile` to get a list of attributes to present in the form.

First we need to declare the service home:

```
private PortletServiceHome lsh;
```

Then we retrieve that home in the init method:

```
if(lsh == null) {
javax.naming.Context ctx = new javax.naming.InitialContext(); lsh
= (PortletServiceHome)
ctx.lookup("portletservice/com.ibm.portal.portlet.service.model.Languag
eListProvider");
}
```

Finally, we add this method:

```
private ArrayList getLanguages(PortletRequest request) {
ArrayList languageList = new ArrayList();
Locale locale = request.getLocale();
boolean serviceAvailable = false;
```

PUMA scenarios

```
try {
if (lsh!= null) {
serviceAvailable = true;
}
if (serviceAvailable) {
LanguageListProvider provider = (LanguageListProvider)
lsh.getPortletService(LanguageListProvider.class);
LanguageList list = provider.getLanguageList(request);
Iterator it = list.iterator();
LanguageBean langBean = null;
Language language = null;
while (it.hasNext()) {
language = (Language)it.next();
langBean = new LanguageBean();
langBean.setLocale(language.getLocale()+"");
langBean.setDescription(language.getTitle(locale));
languageList.add(langBean); }
} catch(Exception ex) {
System.err.println("There was an error retrieving the languages "
+ ex);
}
return languageList;
}
```

Then we place that list into the request so that it is available for the

```
request.setAttribute("languages", getLanguages(request));
```

After that we can create the form on the AddUser.jsp:

```
<portlet:actionURL var="addUserUrl">
<portlet:param name="action" value="addUser" />
</portlet:actionURL>
<form action="<%=addUserUrl %>" method="POST" name="userInfoForm">

<table><tr>
    <td align="right">* </td>
    <td align="left" class="wpsEditText">
        <label for="uid">User ID: <span
style="display:none">required</span> </label>
    </td>
</tr>
<tr>
    <td>&nbsp;</td>
    <td><input dir="ltr" class="wpsEditField" type="text" id="uid"
name="attr_uid" value='' ></td>
    <td class="wpsFieldErrorText"></td>
</tr>
<tr>
    <td align="right">* </td>
    <td align="left" class="wpsEditText"><label
for="userPassword">Password:</label></td>
</tr>
<tr>
```

PUMA scenarios

```

        <td>&nbsp;</td>
        <td><input dir="ltr" class="wpsEditField" type="password"
id="userPassword" name="attr_userPassword" value='' ></td>
        <td class="wpsFieldErrorText">
        </td>
    </tr>
    <tr>
        <td align="right"> * </td>
        <td align="left" class="wpsEditText"><label
for="wps.portlets.confirm_password">Confirm Password:</label></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input dir="ltr" class="wpsEditField" type="password"
id="wps.portlets.confirm_password"
name="attr_wps.portlets.confirm_password" value='' ></td>
        <td class="wpsFieldErrorText"></td>
    </tr><tr>
        <td align="right"></td>
        <td align="left" class="wpsEditText"><label
for="givenName">First Name:</label></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input class="wpsEditField" type="text" id="givenName"
name="attr_givenName" value='' ></td>
        <td class="wpsFieldErrorText"></td>
    </tr>
    <tr>
        <td align="right">*</td>
        <td align="left" class="wpsEditText"><label for="sn">Last
Name:<span style="display:none">required</span></label></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input class="wpsEditField" type="text" id="sn"
name="attr_sn" value='' ></td>
        <td class="wpsFieldErrorText"></td>
    </tr>
    <tr>
        <td align="right"></td>
        <td align="left" class="wpsEditText"><label for="ibm-
primaryEmail">Email:</label></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input dir="ltr" class="wpsEditField" type="text" id="ibm-
primaryEmail" name="attr_ibm-primaryEmail" value='' ></td>
        <td class="wpsFieldErrorText"></td>
    </tr>
    <tr>
        <td align="right"></td>
        <td align="left" class="wpsEditText"><label
for="preferredLanguage">Preferred language:</label></td>
    </tr>
    <tr>
        <td>&nbsp;</td>

```

PUMA scenarios

```

        <td><select id="preferredLanguage"
name="attr_preferredLanguage">
<option value='' selected > - Nothing Selected -</option>
<% ArrayList languages = (ArrayList)request.getAttribute("languages");
if(languages != null) {
LanguageBean langBean = null;
for (int i = 0; i < languages.size(); i++) {
langBean = (LanguageBean)languages.get(i);%>
<option value='<%=langBean.getLocale() %>'
><%=langBean.getDescription() %></option>
<%}
}%>
</select>
</td>
        <td class="wpsFieldErrorText"></td>
    </tr>
    <tr>
        <td class ="wpsEditSmText" colspan="3">* Required Field</td>
    </tr>
    <tr>
        <td colspan="3"><div class="portlet-separator"></div></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>
            <table border="0" cellpadding="0" cellspacing="4">
                <tr>
                    <td nowrap><input class="wpsButtonText"
style="cursor:hand" type="submit" border="0" align="absmiddle"
name="ok" value="OK" />
                    </td>
                    <td nowrap><input class="wpsButtonText"
style="cursor:hand" type="cancel" border="0" align="absmiddle"
name="cancel" value="Cancel" />
                    </td>
                </tr>
            </table>
        </td>
    </tr>
</table>
</form>

```

Now that the form has been submitted, it goes to the processAction method of the portlet. In that method we add a check to determine if it is coming from the addUser form and, if so, we use the PumaController to update the user.

Note that the PumaController requires you to pass in a userid and the map of attributes you want to create for that user. To get the Controller you must have an ActionRequest, which cannot be done in the doView():

```

HashMap userAttrs = new HashMap();
userAttrs.put("uid", request.getParameter("attr_uid"));
userAttrs.put("userPassword",
request.getParameter("attr_userPassword"));
userAttrs.put("givenName", request.getParameter("attr_givenName"));
userAttrs.put("sn", request.getParameter("attr_sn"));

```

PUMA scenarios

```
userAttrs.put("ibm-primaryEmail", request.getParameter("attr_ibm-
primaryEmail"));
userAttrs.put("preferredLanguage",
request.getParameter("attr_preferredLanguage"));
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
try {
pController.createUser(request.getParameter("attr_uid"), null,
userAttrs);
} catch (Exception e) {
System.err.println("the create failed = " + e);
}
```

2.3 Editing a user

In this example we add a link to the display of users so that, once you search for the users, you can click on a pencil icon to edit the attributes of that user.

To create that URL we use the following:

```
<portlet:renderURL var="editUrl">
<portlet:param name="navPage" value="editUser" />
<portlet:param name="uid" value="<%= myUser.getUid()%>" />
</portlet:renderURL>
```

Then in the doView we do the following to retrieve the user attributes and then send them to the edit page for editing:

```
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaProfile pp = service.getProfile(request);
PumaLocator pl = service.getLocator(request);
List userList = pl.findUsersByAttribute("uid",
request.getParameter("uid"));
User editUser = (User)userList.get(0);
Map userMap = pp.getAttributes(editUser, EDIT_USER_ATTRS);
UserViewBean userBean = new UserViewBean();
```

This section of code should be repeated for each item you wish to retrieve; since some items might be multivalued, you must check whether they are arrays or just the String object:

```
Object attrObj = userMap.get("givenName");
ArrayList attrArray = null;
if (attrObj instanceof java.util.List) {
attrArray = (ArrayList)attrObj;
userBean.setGivenName((String)attrArray.get(0));
} else {
userBean.setGivenName((String) attrObj);
}
```

Then, set the object to be pulled by the .jsp for editing:

PUMA scenarios

```
request.setAttribute("editUserBean", userBean);
```

The edit .jsp is quite similar to the add user, except that it pulls the values from the userBean, so we can skip reiterating the form here. One other thing we add to the form is a hidden attribute for the user being edited(uid), so that we can find that same user when doing the update in processAction.

Finally, after submitting the form we are in the processAction and we update the user:

```
HashMap userAttrs = new HashMap();
userAttrs.put("uid", request.getParameter("attr_uid"));
String userPassword = request.getParameter("attr_userPassword");
if(userPassword != null && !userPassword.equals("")) {
userAttrs.put("userPassword", userPassword);
}
userAttrs.put("givenName", request.getParameter("attr_givenName"));
userAttrs.put("sn", request.getParameter("attr_sn"));
userAttrs.put("ibm-primaryEmail", request.getParameter("attr_ibm-
primaryEmail"));
userAttrs.put("preferredLanguage",
request.getParameter("attr_preferredLanguage"));
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
PumaLocator pl = service.getLocator(request);
try {
List userList = pl.findUsersByAttribute("uid",
request.getParameter("attr_uid_hidden"));
User editUser = (User)userList.get(0);
pController.setAttributes(editUser, userAttrs);
} catch (Exception e) {
System.err.println("the create failed = " + e);
}
```

This last piece of code updates the user attributes that have been submitted through the form. You can have any number of attributes to be updated in this way.

2.4 Deleting a user

The third piece of code in dealing with users allows us to delete a user from the system. First we create an action URL that details what user to delete, and what action to perform:

```
<portlet:actionURL var="deleteUrl">
<portlet:param name="action" value="deleteUser" />
<portlet:param name="uid" value="<%= myUser.getId()%>" />
</portlet:actionURL>
```

The delete is handled in the process action via the following code:

```
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
PumaLocator pl = service.getLocator(request);
try {
```

PUMA scenarios

```
List userList = pl.findUsersByAttribute("uid",
request.getParameter("uid"));
User deleteUser = (User)userList.get(0);
pController.deleteUser(deleteUser);
} catch (Exception e) {
System.err.println("there was a problem deleting the user = " + e);;
}
```

2.5 Adding a group

Now we focus on group management. The first method we look at is the ability to add a group. First, we create a link from the main page to add the group:

```
<portlet:renderURL var="newGroup">
<portlet:param name="navPage" value="newGroup" />
</portlet:renderURL>
<tr>
<td class="wpsDialogIconBackground" nowrap="">
<a nowrap="" href="<%=newGroup %>" style="text-decoration: none;">
<img class="wpsDialogIcon" border="0" alt="New Group" title="New Group"
src='<%=renderResponse.encodeURL(renderRequest.getContextPath()+"/image
s/New_Task.gif") %>' />
</a>
</td>
<td class="wpsDialogIconTextBackground" nowrap="" >
<a nowrap="" href="<%=newGroup %>" style="text-decoration: none;">
<span class="wpsDialogIconText"> New Group</span>
</a>
```

Next, we control what page this goes to in the portlet by adding a section in the doView to check for the navPage parameter:

```
else if (pageNav.equals("newGroup")) {
jspName = ADD_GROUP_JSP;
```

In that JSP we have the following form for creating a new group:

```
<form name="newGroupForm" method="POST"
action="<%=addGroupUrl %>"><table width="100%" border="0"
cellpadding="0" cellspacing="5">
<tr>
<td align="left"><span class="portlet-form-label">New
group</span></td>
</tr>
<tr>
<td align="left"><label class="wpsLabelText"
for="newgroupname">ID:</label></td>
</tr>
<tr>
<td align="left"><input class="wpsEditField" type="text"
id="newgroupname"
name="newgroupname"
value="" style="width:350" maxlength="200"></input></td>
</tr>
<tr></tr>
```


PUMA scenarios

```
<tr>
<td align="left"><label class="wpsLabelText"
for="newgroupdesc">Description:</label></td>
</tr>
<tr>
<td align="left"><input class="wpsEditField" type="text"
id="newgroupdesc"
name="newgroupdesc"
value="" style="width:350" maxlength="300"></input></td>
</tr>
</table>
<table border="0" cellpadding="0" cellspacing="4">
<tr>
<td nowrap><input type="submit" name="Ok" value="Ok"></td>
<td nowrap><input type="reset" value="Cancel"></td>
</tr>
</table>
</form>
```

This posts to the portlet, and then in the processAction we retrieve the fields from the form and then use the PumaController to create the group:

```
HashMap groupAttrs = new HashMap();
groupAttrs.put("cn", request.getParameter("newgroupname"));
groupAttrs.put("description", request.getParameter("newgroupdesc"));
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
try {
pController.createGroup(request.getParameter("newgroupname"), null,
groupAttrs);
} catch (Exception e) {
System.err.println("the create of group failed = " + e);
}
```

2.6 Deleting a group

The second method we look at is the ability to add a group. To do this, first we create a link from the main page to delete the group. During the loop to display groups we add a link to delete the group:

```
<portlet:actionURL var="deleteGroupUrl">
<portlet:param name="action" value="deleteGroup" />
<portlet:param name="cn" value="<%= myGroup%>" />
</portlet:actionURL>
```

Then we add a section in the processAction to actually delete this group:

```
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
PumaLocator pl = service.getLocator(request);
try {
```

PUMA scenarios

```
List groupList = pl.findGroupsByAttribute("cn",
request.getParameter("cn"));
Group deleteGroup = (Group)groupList.get(0);
pController.deleteGroup(deleteGroup);
} catch (Exception e) {
System.err.println("there was a problem deleting the group = " + e);;
}
```

2.7 Adding a member to a group

The third step is to add users to the group. First we need to add a link so that we can go from viewing the group membership to adding a user to the group page:

```
<portlet:renderURL var="addMember">
<portlet:param name="navPage" value="addMember" />
<portlet:param name="cn" value='<%=renderRequest.getParameter("cn")
%>' />
</portlet:renderURL>
```

That can be displayed with the following:

```
<td class="wpsDialogIconBackground" nowrap="">
<a nowrap="" href="<%=addMember %>" style="text-decoration: none;">
<img class="wpsDialogIcon" border="0" alt="Add Member" title="Add
Member"
src='<%=renderResponse.encodeURL(renderRequest.getContextPath()+"/image
s/New_Task.gif") %>' />
</a>
</td>
<td class="wpsDialogIconTextBackground" nowrap="" >
<a nowrap="" href="<%=addMember %>" style="text-decoration: none;">
<span class="wpsDialogIconText"> Add Member</span>
</a>
</td>
```

Next in the doView we redirect to the AddMember.jsp:

```
request.getPortletSession().setAttribute("groupToEdit",
request.getParameter("cn"));
jspName = ADD_MEMBER_JSP;
```

This .jsp uses some Ajax to retrieve the users:

```
<script type="text/javascript"
src="<%=request.getContextPath()%>/portletRequest.js">
</script>
<script type="text/javascript">
function <portlet:namespace/>loadFragment(url) {
var searchBy = document.getElementById("searchBy").value;
var queryString = document.getElementById("searchString").value;
url = url + "?searchBy="+escape(searchBy)+"&searchString="+
queryString;
new ibmsample.PortletRequest(url, <portlet:namespace/>refreshDOM);
}
function <portlet:namespace/>refreshDOM() {
var markup = this.request.responseText;
```

PUMA scenarios

```
        document.getElementById("displayTable").innerHTML = markup;
    }
</script>

<form action="">
Search by: <select name="searchBy" id="searchBy">
<option value="uid">uid</option>
<option value="ibm-primaryEmail">ibm-primaryEmail</option>
<option value="sn">sn</option>
</select><br>

Search String: <input type="text" id="searchString" name="searchString"
size="20">
<input type="button" name="Search" value="Search"
onClick="<portlet:namespace/>loadFragment('<%=renderRequest.getContextP
ath()%/>/SearchUsers')">
</form>
<portlet:actionURL var="addMember">
<portlet:param name="action" value="addMember" />
</portlet:actionURL>
<form action="<%=addMember %>">
<p>
<div id="displayTable">
</div>
<p>
<input type="submit" name="Add Member" value="Add Member"> <input
type="reset" value="Cancel">
</form>
<form action="<portlet:actionURL/>">
<input type="submit" name="Back" value="Back">
</form>
```

The Ajax call is calling a servlet that searches PUMA for a list of users matching the search criteria entered in the form. This servlet, however, is not running within the context of an authenticated session, so anonymous access is needed for the virtual resources of users and user groups.

Without the addition of read permission for anonymous users, the code would not be able to retrieve the users in this way:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
try {
    PumaLocator pl = pumaHome.getLocator(request);
    PumaProfile pp = pumaHome.getProfile(request);
    List userList =
    pl.findUsersByAttribute(request.getParameter("searchBy"),
    request.getParameter("searchString"+"*");
    ArrayList newUserList = new ArrayList();
    Map userMap = null;
    UserViewBean newUser = null;
    ArrayList attrArray = null;
    ....
    request.setAttribute("userList", newUserList);
```

PUMA scenarios

```
RequestDispatcher rd =
request.getRequestDispatcher("/_UserManagment/jsp/html/RefreshView.jsp"
);
rd.include(request, response);
} catch (Exception e) {
System.err.println("the users could not be retrieved " +e);
}
}
```

protected void

```
doPost(HttpServletRequest arg0, HttpServletResponse arg1) throws
ServletException, IOException {
doGet(arg0, arg1);
}
```

Finally, the above code redirects to a .jsp that renders the table output of the list, which is dynamically updated into the Add Member form:

```
<%List userList = (List)request.getAttribute("userList"); %>
<%
if(userList != null) { %>
<table border=1> <%
UserViewBean myUser = null;
for (int i = 0; i < userList.size(); i++) {
myUser = (UserViewBean) userList.get(i); %>
<tr>
<td><input type="radio" name="userToAdd" value="<%=myUser.getUId()
%> "/> </td>
<td width="75%">
<%=myUser.getGivenName() %> <%=myUser.getSn()%></td>
</tr>
<% }%>
</table> <%
} else {
%>
No Results were returned
<% } %>
```

When this form is submitted, the user will be added to the group:

```
String userUID = request.getParameter("userToAdd");
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
PumaLocator pl = service.getLocator(request);
try {
List groupList = pl.findGroupsByAttribute("cn",
(String)request.getPortletSession().getAttribute("groupToEdit"));
Group addGroup = (Group)groupList.get(0);
List userList = pl.findUsersByAttribute("uid", userUID);
pController.addToGroup(addGroup, userList);
} catch (Exception e) {
System.err.println("there was an error adding user to group = " + e);
}
```

2.8 Removing a member from a group

This fourth piece also takes place in the section for listing users in a group. First, we create a link to delete the user from the group:

```
<portlet:actionURL var="removeMember"> <portlet:param name="action"
value="removeMember" />
<portlet:param name="cn" value='<%=renderRequest.getParameter("cn")
%>' />
<portlet:param name="uid" value="<%= myUser.getUid()%>"/>
</portlet:actionURL>
<a href="<%=removeMember%>"><img class="wpsDialogIcon" border="0"
alt="Remove User"
src='<%=renderResponse.encodeURL(renderRequest.getContextPath()+"/image
s/remove.gif") %>' /></a>
```

Then in the processAction we actually delete the user:

```
String userUID = request.getParameter("uid");
com.ibm.portal.um.portletservice.PumaHome
service = (com.ibm.portal.um.portletservice.PumaHome)
psh.getPortletService(com.ibm.portal.um.portletservice.PumaHome.class);
PumaController pController = service.getController(request);
PumaLocator pl = service.getLocator(request);
try {
List groupList = pl.findGroupsByAttribute("cn",
request.getParameter("cn"));
Group addGroup = (Group)groupList.get(0);
List userList = pl.findUsersByAttribute("uid", userUID);
pController.removeFromGroup(addGroup, userList);
} catch (Exception e) {
System.err.println("there was an error removing user from group =
"+ e);
}
```

2.9 Customizing password expiration

Many customers need to enforce password expiration. This can be managed from the LDAP server, but along with that you must have WebSphere Portal set up to warn users that their password is about to expire, in accordance with the business rules you have set in place.

In this example we use the pwdChangedTime attribute to calculate the days since the last password change. This attribute is an operational attribute that is returned only if specifically stated in the LDAP request; in addition, it's not available until you've enabled password policy in the LDAP server.

First, you must enable security on an LDAP server of your choice. Follow the steps in the [WebSphere Portal Information Center](#) for enabling security. In this example we use IBM Directory Server (IDS) 5.2, with which the above-mentioned operational attribute is not initially accessible.

After enabling security you must apply a password policy to the IDS server and then restart the service for it to take effect. To do this, we use the following command on the machine running the LDAP Server:

```
C:\>ldapmodify -D cn=root -w rootpassword -i passwordpolicy.txt
```

PUMA scenarios

which generates this output:

```
modifying entry cn=pwdpolicy
C:\>
```

and the passwordpolicy.txt file contains the following:

```
dn: cn=pwdpolicy
changetype: modify
replace: ibm-pwdpolicy
ibm-pwdpolicy: TRUE
#select TRUE to enable, FALSE to disable
-
replace:pwdallowuserchange
pwdallowuserchange: TRUE
#select TRUE to enable, FALSE to disable
-
replace:pwdmustchange
pwdmustchange: TRUE
#select TRUE to enable, FALSE to disable
#Note:If you are running applications that have not been coded
#using the LDAP password policy controls, this policy option is
#not enforced on binds, however no subsequent operations are allowed
#unless the password is changed.
-
replace:pwdsafemodify
pwdsafemodify: TRUE
#select TRUE to enable, FALSE to disable
-
replace:pwdmaxage
pwdmaxage: 7776000
#in seconds 776000=90days
-
replace:pwdexpirewarning
pwdexpirewarning: 86400
#in seconds 86400=1day
-
replace:pwdgracelogleinlimit
pwdgracelogleinlimit: 0
-
replace:pwdMaxFailure
pwdMaxFailure: 5
-
replace:pwdLockoutDuration
pwdLockoutDuration: 30
-
replace:pwdLockout
pwdLockout: TRUE
```

Finally, restart the LDAP server. To verify the password policy is set correctly, run the following `ldapsearch` to see whether the `pwdChangedTimed` is returned:

```
C:\>ldapsearch -D cn=root -w rootpassword -b "uid=joetest5,cn=users,dc=raleigh,dc=ibm,dc=com" objectclass=* pwdChangedTime uid=joetest5,cn=users,dc=raleigh,dc=ibm,dc=com
```

PUMA scenarios

pwdChangedTime=20051115194618.000000Z

Note that this operational attribute must be requested via an admin user.

From the above you can see that the time is in a generalized date format of YYYYMMDDHHMMSS. To make use of this, you must break out the pieces and use them to create a calendar object and then compare to the current date.

For reading and comparing, we created the following utility class:

DISCLAIMER OF WARRANTIES:

The following [enclosed] code is sample code created by IBM Corporation. This sample code is provided to you solely for the purpose of assisting you in the development of your applications. The code is provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample code, even if they have been advised of the possibility of such damages.

```
package com.ibm.password.commands;

import java.io.File;
import java.util.*;

import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.InitialLdapContext;
import javax.xml.parsers.*;

import org.w3c.dom.*;
import org.xml.sax.*;

public class PasswordExpireUtil {

    private String retrieveAttrs(Attributes attrs) {
        String pwdChangedTime = "";
        if (attrs == null) {
            System.out.println("No attributes when trying to print");
        } else {
            /* Print each attribute */
            try {
                for (NamingEnumeration ae = attrs.getAll(); ae.hasMore();) {
                    Attribute attr = (Attribute)ae.next();
                    // System.out.println("attribute: " + attr.getID());
                    if(attr.getID().equals("pwdChangedTime")) {
                        for (NamingEnumeration e = attr.getAll(); e.hasMore();)
                            pwdChangedTime = (String)e.next();
                        break;
                    }
                    /* print each value */
                    //for (NamingEnumeration e = attr.getAll(); e.hasMore();)
                    //System.out.println("value: " + e.next());
                }
            } catch (NamingException e) {
                e.printStackTrace();
            }
        }
    }
}
```

PUMA scenarios

```
}
return pwdChangedTime;
}

public String CheckPassowrdExpiration(String userid) {

    InitialLdapContext ctx = null;
    String INITCTX="com.sun.jndi.ldap.LdapCtxFactory";
    String MY_HOST="ldap://harpy.rtp.raleigh.ibm.com:389";
    String MGR_DN="cn=root";
    String MGR_PW="test123";
    String INITIAL_ENTRY="uid="+userid;
    String pwdChangedTime = "";

    try {
        Hashtable env = new Hashtable();
        env.put(Context.INITIAL_CONTEXT_FACTORY, INITCTX);
        env.put(Context.PROVIDER_URL, MY_HOST);
        env.put(Context.SECURITY_AUTHENTICATION, "simple");
        env.put(Context.SECURITY_PRINCIPAL, MGR_DN);
        env.put(Context.SECURITY_CREDENTIALS, MGR_PW);
        env.put("java.naming.ldap.version", "3");
        ctx = new InitialLdapContext(env, null);

        try {
            // Set up Search Controls
            SearchControls sc = new SearchControls();
            sc.setSearchScope(SearchControls.OBJECT_SCOPE);
            String[] userAttrList = {"sn", "cn", "pwdChangedTime"};
            sc.setReturningAttributes(userAttrList);
            ArrayList nodes = readNodeMaps();
            String userSuffix = "";
            for(int i =0; i < nodes.size(); i++ ) {
                userSuffix = ","+(String)nodes.get(i);
            }
            try {
                NamingEnumeration ne = ctx.search(INITIAL_ENTRY+userSuffix,
                    "(objectclass=*)", sc);
                if(ne!=null){
                    while(ne.hasMore()){
                        SearchResult searchresult = (SearchResult) ne.next();
                        Attributes attrs = searchresult.getAttributes();
                        pwdChangedTime = retrieveAttrs(attrs);
                        //System.out.println("Time Retrieved = " + pwdChangedTime);
                    }
                }else{
                    System.err.println("Search Result is null");
                }
            } catch (NameNotFoundException ex) {
                //System.err.println("Error from search = " + ex);
            }
            } catch (NamingException e) {
                System.err.println("Naming exception on password date lookup " + e);
            }
            ctx.close();
        } catch (Exception f) {
            System.err.println("Connection unable to be opened " + f);
        }
    }
}
```


PUMA scenarios

```
}
return pwdChangedTime;
}

private ArrayList readNodeMaps() {
ArrayList nodes = new ArrayList();
try {
String file = "C:\\WebSphere\\PortalServer\\wmm\\wmm.xml";
DocumentBuilderFactory
docBuilderFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
Document doc = docBuilder.parse(new File(file));
doc.getDocumentElement().normalize();
NodeList listOfNodes = doc.getElementsByTagName("nodeMap");
Element nodeMap = null;
for(int i =0; i < listOfNodes.getLength(); i++ ) {
nodeMap = (Element)listOfNodes.item(i);
nodes.add(nodeMap.getAttribute("node"));
// System.out.println("node from wmm.xml = " +
nodeMap.getAttribute("node"));
}

} catch(SAXParseException err) {
System.out.println("** Parsing error, line " + err.getLineNumber()
+ ", uri " + err.getSystemId());
System.out.println(" " + err.getMessage());
} catch(SAXException e) {
Exception x = e.getException();
System.err.println(x);
} catch(Throwable t) {
}
return nodes;
}

public int getDays(GregorianCalendar g1, GregorianCalendar g2) {
int elapsed = 0;
Calendar gc1, gc2;

if (g2.after(g1)) {
gc2 = (GregorianCalendar) g2.clone();
gc1 = (GregorianCalendar) g1.clone();
}
else {
gc2 = (GregorianCalendar) g1.clone();
gc1 = (GregorianCalendar) g2.clone();
}

gc1.clear(Calendar.MILLISECOND);
gc1.clear(Calendar.SECOND);
gc1.clear(Calendar.MINUTE);
gc1.clear(Calendar.HOUR_OF_DAY);

gc2.clear(Calendar.MILLISECOND);
gc2.clear(Calendar.SECOND);
gc2.clear(Calendar.MINUTE);
gc2.clear(Calendar.HOUR_OF_DAY);

while ( gc1.before(gc2) ) {
```

PUMA scenarios

```
        gc1.add(Calendar.DATE, 1);
        elapsed++;
    }
    return elapsed;
}
}
```

The first public method takes in a string object that is the UID and then returns the string from the `pwdChangedTime`. This method searches all the node maps as defined in the `wmm.xml` for WebSphere Portal and searches for the user in those trees. This way, whenever you have a change in `wmm.xml`, it will pull the user from wherever in the tree is necessary.

To make this code work in your environment, you must change the file location of the `wmm.xml` file in the method `readNodeMaps`, as well as these connection properties from `CheckPasswordExpiration()`:

```
String MY_HOST="ldap://harpy.rtp.raleigh.ibm.com:389";
String MGR_DN="cn=root";
String MGR_PW="test123";
String INITIAL_ENTRY="uid="+userid;
```

The last public method is `getDays`. This method takes in two Gregorian calendar objects and returns the number of days that exist between the two dates.

The previous utility classes are called like this from a v5.1 code perspective. If your password is within five days of expiring, the code will redirect you to a screen displaying a warning message and then directs you to the Change Profile page. To create this, we extend the class `LoginUserAuth`, and then overwrite the `doPostLogin` only:

DISCLAIMER OF WARRANTIES:

The following [enclosed] code is sample code created by IBM Corporation. This sample code is provided to you solely for the purpose of assisting you in the development of your applications. The code is provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample code, even if they have been advised of the possibility of such damages.

```
package com.ibm.password.commands;

import java.io.IOException;
import java.util.GregorianCalendar;

import com.ibm.portal.WpsException;
import com.ibm.wps.engine.RunData;

public class
LoginUserAuth extends com.ibm.wps.engine.commands.LoginUserAuth {

    /* (non-Javadoc)
    * @see
    com.ibm.wps.engine.commands.LoginUser#doPostLogin(com.ibm.wps.engine.Ru
nData, java.lang.String, java.lang.String)
    */
```

PUMA scenarios

```
protected void doPostLogin(RunData rundata, String userid, String
password)
throws WpsException {
super.doPostLogin(rundata, userid, password);
PasswordExpireUtil peUtil = new PasswordExpireUtil();
String pwdTime = peUtil.CheckPassowrdExpiration(userid);
GregorianCalendar rightNow = new GregorianCalendar();
GregorianCalendar pwdCalendar = new GregorianCalendar();
pwdCalendar.set(Integer.valueOf(pwdTime.substring(0,4)).intValue(),
Integer.valueOf(pwdTime.substring(4,6)).intValue()-1,
Integer.valueOf(pwdTime.substring(6,8)).intValue());
int numberOfDays = peUtil.getDays(rightNow, pwdCalendar);
\\System.out.println("number of days that have elapsed = " +
numberOfDays);
if(numberOfDays > 85) {
try {
String screenURL = URLHelper.createScreenURL("PasswordExpire",
rundata.getRequest(), rundata.getResponse(), Boolean.TRUE);
\\System.out.println("doPostLogin: setting redirect to Screen URL = "
+ screenURL);
rundata.setRedirectURL(screenURL);
} catch (IOException e) {
System.err.println("error from login = " + e);
}
}
}
}
```

After compiling and exporting this as a .jar, place it in shared/app under websphere/portalserver. Then update loaderservice.properties so that the following key/value pair includes the package that you placed LoginUserAuth in above:

```
command.path =
com.ibm.password.commands;com.ibm.wps.engine.commands;com.ibm.wps.dynamicui.com
mands
```

To get LoginUserAuth.java to compile, we need to add the following .jars to the build path in rad:

```
J2ee.jar, dynacache.jar, wp.base.jar, wp.engine.cmd.jar, wp.engine.imp.jar, wp.ui.jar,
wp.services.impl.jar, wp.services.api.jar, wp.admin.common.jar, wp.auth.cmd.jar,
wp.model.api.jar
```

One other piece that is needed is a helper class to create the URL for the screen. (This is the same helper class as used in the User Agreement form discussed earlier, so it will not be discussed in detail here.):

```
package com.ibm.password.commands;

import java.io.IOException;
import java.io.StringWriter;

import javax.naming.*;
import javax.servlet.http.*;
```

PUMA scenarios

```
import com.ibm.portal.state.EngineURL;
import com.ibm.portal.state.accessors.action.engine.logout.*;
import com.ibm.portal.state.accessors.screentemplate.*;
import com.ibm.portal.state.accessors.url.URLAccessorFactory;
import com.ibm.portal.state.exceptions.StateException;
import com.ibm.portal.state.service.*;

public class URLHelper {
    /** the JNDI name to retrieve the
    PortalStateManagerServiceHome object */
    private static final String JNDI_NAME =
        "portal:service/state/PortalStateManager";

    /**
    * The PortalStateManagerServiceHome object to
    retrieve the service from
    */
    private static PortalStateManagerServiceHome serviceHome;

    /** Any method processing request and response. */
    public static String createScreenURL(final String screen,
    final HttpServletRequest request, final
    HttpServletResponse response, Boolean protectedURL)
    throws IOException {

        try {
            // get the service from our home interface
            final PortalStateManagerService service =
                getServiceHome().getPortalStateManagerService(request, response);

            // get a URL from the URL accessor factory using request and
            response
            final URLAccessorFactory urlFct = (URLAccessorFactory)
                service.getAccessorFactory(URLAccessorFactory.class);

            // the URL should be based on the current request state
            final EngineURL url = urlFct.newURL(request, response,
                com.ibm.portal.state.Constants.SMART_COPY);

            url.setProtected(protectedURL);

            // change the screen
            final ScreenTemplateAccessorFactory screenFct =
                (ScreenTemplateAccessorFactory)
                service.getAccessorFactory(ScreenTemplateAccessorFactory.class);
            final ScreenTemplateAccessorController screenCtrl =
                screenFct.getScreenTemplateAccessorController(url.getState());
            screenCtrl.setScreenTemplate(screen);
            screenCtrl.dispose();

            // indicate that we do not need the service any longer
            service.dispose();

            // stream the URL using the given writer
            return url.writeDispose(new StringWriter()).toString();
        }
    }
}
```

PUMA scenarios

```
    } catch (StateException e) {
        // error handling
    }

    return "";
}

public static String createLogoutURL(final HttpServletRequest request,
final HttpServletResponse response) throws IOException {

    try {
        // get the service from our home interface
        final PortalStateManagerService service =
getServiceHome().getPortalStateManagerService(request, response);

        // get a URL from the URL accessor factory using request and
response
        final URLAccessorFactory urlFct = (URLAccessorFactory)
service.getAccessorFactory(URLAccessorFactory.class);

        // the URL should be based on the current request state
        final EngineURL url = urlFct.newURL(request, response,
com.ibm.portal.state.Constants.SMART_COPY);

        // set the logout action
        final LogoutActionAccessorFactory logoutFct =
(LogoutActionAccessorFactory)
service.getAccessorFactory(LogoutActionAccessorFactory.class);
        LogoutActionAccessorController
logoutCtrl=logoutFct.newLogoutActionController(url.getState());

        //do not need the controller any more
        logoutCtrl.dispose();

        // indicate that we do not need the service any longer
        service.dispose();

        // stream the URL using the given writer
        return url.writeDispose(new StringWriter()).toString();

    } catch (StateException e) {
        // error handling
    }

    return "";
}

/**
 * Looks up the PortalStateManagerServiceHome being valid
 * for the lifetime of the portal.
 */
private static PortalStateManagerServiceHome getServiceHome() {
    if (serviceHome == null) {
        try {

            final Context ctx = new InitialContext();
```

PUMA scenarios

```

        serviceHome = (PortalStateManagerServiceHome)
ctx.lookup(JNDI_NAME);
    } catch (Exception e) {
        // error handling
    }

}
return serviceHome;
}
}

```

The screen that's used to show the warning message is shown in the code below and is placed in /websphere/appserver/installedapps/servername/wps.ear/wps.war/screens/html. The file name is PasswordExpire.jsp, and must be this name so that it matches what is requested above in the LoginUserAuth class we created:

```

<%@ page session="false" buffer="none" %>
<%@ taglib uri="/WEB-INF/tld/portal.tld" prefix="wps" %>
<%@ taglib uri="/WEB-INF/tld/portal-internal.tld" prefix="wps-
internal" %>
<wps-internal:adminNavHelper/>
<wps:constants/>
<wps:defineObjects/>
<%@ include file="BidiInclude.jsp" %>
<%-- Licensed Materials - Property of IBM, 5724-
E76, (C) Copyright IBM Corp. 2001, 2004 - All Rights reserved. --%>
<%@ page import="com.ibm.portal.*"%>

<%@ page import="com.ibm.wps.portlets.admin.AdminPortletUtils,
com.ibm.wps.util.CreateUrlCommand,
com.ibm.wps.portlets.admin.shared.*"%>
<TABLE border="0" width="100%" height="400">
<TBODY>
<TR>
<TD>
                <CENTER><h2>Password Expiring Notice</h2>
                <p>
                Your Passowrd is about to expire please click
the link below and change it now.</p>
                <p>
                <wps-internal:adminlinkinfo
name="<%=AdminUniqueNamesMappingService.SELFCARE%>">
<wps:urlGeneration contentNode="<%=wpsContentNode%>"
compositionNode='<%= wpsCompositionNode %>'
portletWindowState="Normal">
                    <wps:urlParam type="render"
name="<%= SharedParamConstants.ACCESS_ORIGIN %>"
value="<%= SharedParamConstants.THEME %>"/>
                    <wps:urlParam type="render" name="<%=
SharedParamConstants.ORIGIN_CONTENT_NODE %>" value="<%=
wpsContentNodeID %>" />
                    <a href="<%= wpsURL.write(escapeXmlWriter); %>"
class="wpsToolBarLink">Change Password</a>
                </wps:urlGeneration>
                </wps-internal:adminlinkinfo>

```

PUMA scenarios

If you do not click the link the page will automatically forward there in 10 seconds.

```
                </p></CENTER>
            </TD>
</TR>
</TBODY>
</TABLE>
</form>
```

2.9.1 Alternate method for retrieving the data

Instead of using direct LDAP calls, if the properties are mapped correctly in WMM, you can retrieve these using PUMA. In this next example we modify the password utils class to use PUMA instead of making LDAP calls by replacing the checkPassword method with the following:

```
public String checkPasswordExpiration(String userid) {
String pwdTime = "";
if(pHome ==null) {
try {
Context ctx = new InitialContext();
Name myjndiname = new
    CompositeName(PumaHome.JNDI_NAME);
pHome = (PumaHome) ctx.lookup(myjndiname);
} catch (Exception ex) {
System.err.println("There was a problem retrieving
the PumaHome = " + ex);
}
}
if(pHome != null) {
PumaLocator pLocator = pHome.getLocator();
try {
List userList = pLocator.findUsersByAttribute("uid", userid);
User user = (User)userList.get(0);
System.out.println("This is the user " + user);
PumaProfile pProfile = pHome.getProfile();
List attrs = new ArrayList();
attrs.add("pwdChangedTime");
attrs.add("uid");
Map userInfo = pProfile.getAttributes(user, attrs);
Iterator it = userInfo.entrySet().iterator();
while (it.hasNext()) {
    Map.Entry pairs = (Map.Entry)it.next();
    System.out.println(pairs.getKey() + " = " + pairs.getValue());
}
pwdTime = (String)userInfo.get("pwdChangedTime");
} catch (Exception ex) {
System.err.println("there was an exception getting the user = " + ex);
}
}
return pwdTime;
}
```

3 Implementation scenario: User Agreement use case

This section explains how to create a User Agreement use case that forces users to sign before they can access the protected pages.

3.1 User Agreement form

A common site requirement is that users must sign an agreement before being allowed access to protected resources and pages. Although WebSphere Portal does not provide this mechanism out of the box, it does provide all the necessary components to implement this feature.

There are several different methods and design approaches to produce a User Agreement form. We discuss the pros and cons of the preferred methods and the implications of using the other methods.

The User Agreement implementation we choose requires several different components to work effectively:

- A custom LoginUserAuth implementation

NOTE: Regarding the LoginUserAuth class throughout this document, if you have a WebSphere Portal environment in which security is not enabled, you will need to use the LoginUserNoAuth class rather than LoginUserAuth.

- A custom log-in portlet
- URL generation APIs

Moreover, we must consider the following factors when architecting and developing a User Agreement scenario:

- When the user logs in, we need a way to determine whether they have signed the agreement previously, or had signed the agreement within a given period.
- If the user has not signed the agreement, we need to redirect them to a page on which they can review and sign the agreement.
- If the user accepts the agreement, then they should be allowed to authenticate.
- If the user refuses the agreement, then they should be prevented from authenticating.

In this scenario, the process flow would be as follows:

1. User navigates to the WebSphere Portal log-in page.
2. User enters their credentials and attempts to log in.
3. The log-in portlet kicks off the log-in process.
4. The log-in determines if the user has signed the agreement.
5. If the agreement has been signed, the user is allowed to authenticate
6. If the agreement has not been signed, the user is redirected back to the log-in page, and they are given the opportunity to sign the agreement. Repeat starting at step 2.

We will describe our implementation based on the flow listed above. We do not provide step-by-step instructions on creating each file; instead we provide details on the relevant implementation code. The sample code is provided as a download, along with some of the snippets, which you can reference for more information.

3.2 Custom Log-in portlet

The first component we must to implement is the log-in portlet. Starting in WebSphere Portal version 6.0.1, a log-in service is provided to allow for the creation of a custom log-in portlet. This log-in service is available only for JSR 168 portlets.

Create a new portlet and add the following code to the portlet class, which shows how to obtain the log-in service by using a JNDI lookup:

Fist, create a class variable to hold the LoginHome object:

```
private LoginHome loginHome = null;
public void init() throws PortletException{
super.init(); try {
PortletServiceHome psh;
javax.naming.Context ctx = new javax.naming.InitialContext();
psh = (PortletServiceHome) ctx.lookup(LoginHome.JNDI_NAME);
loginHome = (LoginHome) psh.getPortletService(LoginHome.class);
} catch(Exception ex) {
System.out.print("The services could not be loaded = " + ex);
}
}
```

Since a JNDI lookup can be costly, you should perform this operation in the portlet's init method because it is called only once. The processAction method of the portlet is where we test for valid responses from the user and perform the log-in. The following is the complete processAction code (we will discuss the details later):

```
public void processAction(ActionRequest request, ActionResponse
response) throws PortletException, java.io.IOException {
if( request.getParameter(FORM_SUBMIT) != null ) {
LoginService loginService = (LoginService)
loginHome.getLoginService(request, response);
String userId = request.getParameter(FORM_ID);
String password = request.getParameter(FORM_PASSWORD);
String agreement = request.getParameter(FORM_RADIO_GROUP);
System.out.println("Agreement : " + agreement);
boolean proceed = true;
if(agreement != null){
//this means there was a error
boolean agreementBool = Boolean.valueOf(agreement).booleanValue();
System.out.println("Agreement to a bool: " + agreementBool);
if(!agreementBool){
//fail
proceed = false;
response.setRenderParameter(Constants.USER_AGREEMENT_ERROR_KEY,
"You must agree to the conditions before you're able to login");
} else {

//set the agreementIUserAgreement ua =
SingletonUserAgreement.getInstance();System.out.println("setting
agreement to true");

ua.setUserAgreement(userId, true);
```

PUMA scenarios

```
}
}
//only continue if they have agreed
if(proceed){
Map contextMap = new HashMap();
contextMap.put(LoginService.DO_RESUME_SESSION_KEY, new Boolean(false));
try {
loginService.login(userId, password.toCharArray(), contextMap, null);
} catch (Exception ex) {
System.out.println("this login failed with = " + ex);
//set render parameter to display error.
} finally {
System.out.println("I am in the finally");
}
}
}
}
}
```

This first test in the processAction determines whether the action request is sent by the log-in form. If it is, we obtain an instance of the LoginService to be used later to initiate to log-in process.

Note that we also check for a FORM_RADIO_GROUP request parameter, which is used later when we're processing the user agreement. For now the condition block that tests for this parameter can be ignored.

The next condition for which we test is the proceed Boolean. By default we set this value to true, so the processing should continue.

The last step in the processAction is to log in the user. Notice that the loginService.login method requires a map as one of the arguments. This map must contain an entry with a key of LoginService.DO_RESSUME_SESSION_KEY and a Boolean value.

Below is the JSP source code containing the log-in form for reference, since this also needs to be contained in the log-in portlet:

```
<%@page session="false" contentType="text/html" pageEncoding="ISO-8859-1" import="java.util.*, javax.portlet.*, sampleloginportlet.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<%@page import="com.ibm.wps.l2.useragreement.Constants"%>
<%@page import="com.ibm.wps.l2.useragreement.IUserAgreement"%>
<%@page
import="com.ibm.wps.l2.useragreement.impl.SingletonUserAgreement"%>
<portlet:defineObjects/>
```

```
<%
sampleloginportlet.SampleLoginPortletSessionBean sessionBean =
(sampleloginportlet.SampleLoginPortletSessionBean)renderRequest.getPort
letSession().getAttribute(sampleloginportlet.SampleLoginPortlet.SESSION
_BEAN);
%>
```

```
<%
//handle case where Login came from somewhere else (the Login URL)
boolean needAgreement = false;
```

PUMA scenarios


```
String error =
renderRequest.getParameter(Constants.USER_AGREEMENT_ERROR_KEY);
if(error != null){
    out.println(error + "<br>");
    needAgreement = true;
}%>
<DIV style="margin: 12px; margin-bottom: 36px">
<% /***** Start of sample code *****/ %>

<FORM method="POST" action="<portlet:actionURL/>">
    <% if(needAgreement) { %>
        Do you agree to the terms<br>
        Yes: <input type="radio"
name="<%= SampleLoginPortlet.FORM_RADIO_GROUP %>" value="true">
        No: <input type="radio"
name="<%= SampleLoginPortlet.FORM_RADIO_GROUP %>" value="false">
        <br><br>
    <% } %>
<LABEL
for="<%=sampleloginportlet.SampleLoginPortlet.FORM_ID%>">User:</LABEL><
BR>
<INPUT name="<%=sampleloginportlet.SampleLoginPortlet.FORM_ID%>"
type="text"/><br>
<LABEL
for="<%=sampleloginportlet.SampleLoginPortlet.FORM_PASSWORD%>">Password
:</LABEL><BR>
<INPUT name="<%=sampleloginportlet.SampleLoginPortlet.FORM_PASSWORD%>"
type="password"/>
<br>
Clear agreements <input type="checkbox" name="<%=
SampleLoginPortlet.CLEAR_AGREEMENTS %>" value="<%=
SampleLoginPortlet.CLEAR_AGREEMENTS %>">
<INPUT name="<%=sampleloginportlet.SampleLoginPortlet.FORM_SUBMIT%>"
type="submit" value="Submit"/>
</FORM>
<% /***** End of sample code *****/ %><br>TEST PURPOSE ONLY

</DIV>
```

Figure 4 illustrates how the log-in portlet looks when a user first accesses the log-in page.

Figure 4. Sample login portlet



The screenshot shows a web portlet titled "SampleLoginPortlet". It contains a login form with the following elements:

- A label "User:" followed by a text input field.
- A label "Password:" followed by a password input field.
- A "Submit" button below the password field.

3.3 Determining whether a user has signed the agreement

We felt the best place to determine whether the user has signed the agreement is in the LoginUserAuth class. If the logic for detecting whether the user has signed the agreement were placed only in the log-in portlet, the user could bypass the test by simply calling the log-in URL in a browser passing in the userid and password.

Placing the logic in the log-in portlet would also not work if you use the log-in screen. We could have also opted to place this code in the themes; however, that would require updates to all existing themes, resulting in code that is not very maintainable.

The LoginUserAuth class is called whenever a user attempts to log in and whenever the session times out. This is important because there are a few different mechanisms by which you can log into WebSphere Portal; that is, the log-in portlet, the log-in screen, and the log-in URL. The LoginUserAuth class covers all three of these cases.

Within the LoginUserAuth class are a few different options to consider; for example, you could authenticate the user first and then determine if they have signed, or you could check before the user is authenticated.

For simplicity, our implementation for checking and storing the user agreement information is stored in a hashmap within a singleton. The user ID is the key in the map. We chose this method because the user ID is available to us in the doPreLogin method of the LoginUserAuth, allowing us to test if the user has agreed before logging them in.

As an alternative, you could store the agreement information in LDAP and still use the user's ID, but you would need to update the Virtual Resources to allow additional privileges for anonymous users.

NOTE: Our approach is not without flaws; however, for this demonstration we traded simplicity for robustness. Since the check for the agreement is performed before the user is authenticated, you could enter another user name in the log-in portlet with an arbitrary password. The log-in process would redirect to the log-in portlet, giving the user the chance to agree to the terms. So, when the "real" user logs in, they are not prompted for the agreement.

To avoid this problem you could move the code from the doPreLogin method to the doPostLogin method of the LoginUserAuth class. You would then need to log out the user, if they had not signed the agreement.

3.3.1 SingletonUserAgreement

As stated above, the LoginUserAuth uses the SingletonUserAgreement, and the implementation uses a HashMap to store the agreement information. The hasUserSignedAgreement method simply looks in the map for this user and tests whether they have signed the agreement:

```
public class SingletonUserAgreement implements IUserAgreement {

    private static Map userAgreementMap = new HashMap();

    private static SingletonUserAgreement singletonUserAgreement
    = null;

    private SingletonUserAgreement() {

    }
}
```

PUMA scenarios

```
public static SingletonUserAgreement getInstance() {
    if (singletonUserAgreement == null) {
        singletonUserAgreement = new SingletonUserAgreement();
    }
    return singletonUserAgreement;
}

public boolean hasUserSignedAgreement(String userID) {

    // look for userID key in MAP
    Boolean hasSigned = (Boolean) userAgreementMap.get(userID);

    if(hasSigned != null){

        return hasSigned.booleanValue();
    }

    return false;
}

public void setUserAgreement(String userID, boolean agreed) {

    if (agreed) {
        userAgreementMap.put(userID, new Boolean(agreed));
    }

}

}
```

The doPreLogin of the LoginUserAuth class is called before the user is authenticated. We use this method for checking the user agreement:

```
protected void doPreLogin(com.ibm.wps.engine.RunData runData,
String userID, String password) throws com.ibm.portal.WpsException {
if (userID == null) {
try {
Subject wasSubject = WSSubject.getCallerSubject();
if (wasSubject != null) {
Iterator credentials = wasSubject.getPublicCredentials(
WSCredential.class).iterator();
if (credentials != null) {
if (credentials.hasNext()) {
WSCredential wsCredential = (WSCredential) credentials.next();
userID = wsCredential.getSecurityName();
}
}
}
} catch (CredentialExpiredException e) {
e.printStackTrace();
}
```

PUMA scenarios

```
} catch (CredentialDestroyedException e) {
e.printStackTrace();
} catch (WSSecurityException e) {
e.printStackTrace();
}
}
}

IUserAgreement userAgreement = SingletonUserAgreement.getInstance();
if (!userAgreement.hasUserSignedAgreement(userID)) {
// this should cause the onAuthenticationError to be called passing
// in an ErrorBean with a value of 1
throw new UserAgreementException(
"User has not signed the agreement");
}
System.out.println("Exit LoginUserAuth.doPreLogin");
}
```

In the event of a session timeout, the userID passed to the doPreLogin will be null. To handle this, we obtain the userID from the WSCredential object.

Once we have a userID, we obtain an instance of our SingletonUserAgreement and then test whether the user has signed the agreement. If they have not, we throw a UserAgreementException, resulting in the onAuthenticationError method of the LoginUserAuth class to be invoked. From there we can handle the redirect back to the log-in portlet to give the user the opportunity to sign the agreement.

3.4 Redirecting the user to the Agreement page

When an exception is thrown in the doPreLogin, the onAuthenticationError method will be called. Since we still want to handle all other types of errors during log in, we first need to test for our custom error:

```
if (errorBean.getErrorCode() == OTHER_ERROR) {
// assume we failed as a result of the not signed user agreement

String targetURL = generateURL(runData.getRequest(),
runData.getResponse());
if (targetURL != null) {
runData.setRedirectURL(targetURL);
} else {
// creation of the url failed so let the normal process take over
super.onAuthenticationError(runData, errorBean);
}
// redirect to the Login page pass a parameter to inform the
} else {
super.onAuthenticationError(runData, errorBean);
}
}
```

where "OTHER_ERROR" has a value of 1.

If the error code is not 1 or we fail when trying to generate the URL, we want the default implementation of the onAuthenticationError to execute.

The call to generateURL uses the URLHelper package to generate a stateless URL to the log-in portlet. The URL also passes a parameter to the log-in portlet to indicate that an error occurred, allowing us to display the user agreement in the portlet:

```
private String generateURL(HttpServletRequest request,
    HttpServletResponse response) {
    try {
        HashMap map = new HashMap();
        String[] value1 = { Constants.AGREEMENT_NOT_SIGNED };
        MyServerContext serverContext = new
        MyServerContext(request.getServerName(),
        Integer.toString(request.getServerPort()));
        map.put(Constants.USER_AGREEMENT_ERROR_KEY, value1);
        String url = OffLineURLHelper.generateUrl("wps.Login",
        "myCustom.Login.portlet", map, serverContext, false);
        return url;
    } catch (StateException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

3.5 Displaying the User Agreement form in the log-in portlet

In the portlets view JSP we first check for the parameter set before we generate the URL. If the parameter is present, that means this portlet is being called as the result of a user not having signed the agreement yet:

```
boolean needAgreement = false;
String error =
renderRequest.getParameter(Constants.USER_AGREEMENT_ERROR_KEY);
if(error != null){
out.println(error + "<br>");
needAgreement = true;
}
```

We set a flag that will be used later to display the User Agreement form.

NOTE: As an alternative, we could have picked up the parameter in the doView method of the portlet and redirected to a different JSP, if the agreement needed to be signed. We chose the above approach for simplicity.

If the flag is set and the agreement needs to be signed, then we display the User agreement form:

```
<% if(needAgreement) { %>
Do you agree to the terms<br>
Yes: <input type="radio" name="<%=
SampleLoginPortlet.FORM_RADIO_GROUP %>" value="true">
No: <input type="radio" name="<%=
SampleLoginPortlet.FORM_RADIO_GROUP %>" value="false">
<br>
```

<% } %>

Figure 5 shows the User Agreement form in the log-in portlet.

Figure 5. User Agreement form

SampleLoginPortlet

Agreement not signed

Do you agree to the terms

Yes: No:

User:

Password:

Submit

This brings us back to the processAction method of the log-in portlet. Now, when the user submits the log-in form, the FORM_RADIO_GROUP parameter will be present, and the following code in processAction will be executed:

```

if(agreement != null){
    //this means there was a error
    boolean agreementBool = Boolean.valueOf(agreement).booleanValue();
    System.out.println("Agreement to a bool: " + agreementBool);
    if(!agreementBool){
        //fail
        proceed = false;
        response.setRenderParameter(Constants.USER_AGREEMENT_ERROR_KEY,
            "You must agree to the conditions before you're able to login");
    } else {
        //set the agreement
        IUserAgreement ua = SingletonUserAgreement.getInstance();
        System.out.println("setting agreement to true");
        ua.setUserAgreement(userId, true);
    }
}

```

The code above determines whether the user has signed the agreement. If the user has not signed the agreement, the log-in process is aborted, and the view mode of the portlet will be executed. In the JSP we display a message stating that the user must sign the agreement before they're allowed to log in. If the user agrees, we add them to the SingletonUserAgreement.

In figure 6 you can see that the user elected to not agree with the user agreement, and they were redirected to the log-in form rather than being authenticated.

Figure 6. Redirect to the log-in form

SampleLoginPortlet

You must agree to the conditions before you're able to login

Do you agree to the terms

Yes: No:

User:

Password:

Submit

4 Conclusion

This paper has explained how to architect and create various use cases in using PUMA, including a User Management portlet, Password Expiration warning, and User Agreement. The scenarios provided lay down a foundation that can be extended or modified to meet most business needs around user management and customization.

Many developers want to know how they can leverage the PUMA SPI to create custom log-in portlet and log-in scenarios. In each case we have documented how the PUMA interface can be used to extract and update information on the users in log-in and user management. One example demonstrates how you can do this by using the existing log-in flow with the addition of the URL generation APIs.

5 Resources

IBM WebSphere Portal 6.0 Information Center:

<http://publib.boulder.ibm.com/infocenter/wpdoc/v6r0/index.jsp>

About the Unicode standard:

<http://www.unicode.org/standard/standard.html>

URL Generation SPI Helpers:

<http://www-1.ibm.com/support/docview.wss?rs=688&uid=swg21265900>

Creating a custom user registration portlet using WebSphere Portlet Factory V6:

http://www.ibm.com/developerworks/websphere/library/techarticles/0706_lukito/0706_lukito.html

6 About the authors

James Barnes is a Team Lead for the WebSphere Portal Level 2 Support organization. He focuses on API issues with WebSphere Portal and has written or co-written other publications on theme development as well as the [IBM WebSphere Portal for Multiplatforms V5.1 Handbook](#). He can be reached at jwbarnes@us.ibm.com.

Thomas Hurek is a software architect at the IBM development lab in Germany. He is responsible for technically programming/validating Fix Packs and works as a Consultant in the WebSphere Portal Lab-based Services Team. You can contact Thomas at thomas_hurek@de.ibm.com.

PUMA scenarios

Ryan Wilson is the Technical Lead for the WebSphere Portal Level 2 API/Migration team in RTP, North Carolina. His areas of expertise include J2EE application development with IBM WebSphere Studio and IBM Rational® Application Developer. He has participated in many projects, including developing internal tools and co-authoring other works including the [IBM Rational Application Developer V6 Portlet Application Development and Portal Tools](#) Redbooks® publication. Ryan holds certifications in WebSphere Portal development (5.1, 6.0), and is a Java Certified Programmer and Java Certified Web Component Developer.

Trademarks

- IBM, Rational, Redbooks, and WebSphere are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Other company, product, and service names may be trademarks or service marks of others.