

Setting up a custom user repository with Virtual Member Manager for IBM® WebSphere® Application Server 6.1 and IBM WebSphere Portal 6.1

Andreas Zehnpfenning

IBM Software Group
WebSphere Portal Development
Boeblingen, Germany

Stefan Schmitt

IBM Software Group
WebSphere Portal Security Architect
Boeblingen, Germany

Jan-Paul Buchwald

IBM Software Group
WebSphere Portal Lab Services
Boeblingen, Germany

April 2009

© Copyright International Business Machines Corporation 2009. All rights reserved.

Abstract: Starting with version 6.1, IBM® WebSphere® Application Server leverages a component called Virtual Member Manager (VMM) to manage information about users, user profiles, and user groups. VMM offers a specific Service Provider Interface (SPI), *com.ibm.wsspi.wim.Repository*, to connect VMM with user repositories that VMM does not support out of the box. This white paper explains how to implement a *Repository* instance as part of a custom WebSphere Portal configuration.

Contents

1 Introduction	3
2 WebSphere Portal's use of VMM	3
3 VMM Overview	5
3.1 <i>Types of information managed by VMM</i>	5
3.1.1 Authentication information	5
3.1.2 Group membership	6
3.1.3 User profile	7
3.1.4 Property Extension repository	7
3.2 <i>VMM concepts</i>	8
3.2.1 VMM functions	8
3.2.2 DataObjects	9
3.2.3 Virtual Member Manager unique identifiers	9
3.2.4 Attributes in VMM	10
3.3 <i>Mapping user profile attributes to the user repository</i>	11
3.4 <i>Configuration files for VMM</i>	11
3.4.1 Main configuration file: wimconfig.xml	11
3.4.2 Additional custom attributes: wimxmlextension.xml	13
4 Implementing a custom repository	13
4.1 <i>Comparing WMM and VMM</i>	13
4.2 <i>Methods of the Repository interface</i>	15
4.2.1 Methods for retrieving entities	15
4.2.2 Creating new entities	19
4.2.3 Deleting entities	20
4.2.4 Getting meta data	21
4.2.5 Logging in	22
4.2.6 Searching for entities	23
4.2.7 Updating attributes and membership information	24
4.2.8 Initializing the repository	27
4.2.9 Creating a new attribute definition	28
5 Setting up a custom repository	28
5.1 <i>Using the custom repository within a federated repository</i>	28
5.1.1 Add custom repository as additional federated repository	28
5.1.2 Assign administrator role to user in custom repository (optional)	30
5.1.3 Delete other repositories from VMM configuration (optional)	30
5.2 <i>Using the custom repository as a standalone registry</i>	31
6 Tips and tricks	32
Constants	32
Utility classes	32
7 Verifying your implementation	33
8 The File Repository sample adapter	33
8.1 <i>Architecture</i>	33
9 Conclusion	34
10 Resources	35
11 About the authors	35

1 Introduction

In this paper, the term “custom user repository” refers to any user or group data store that is not natively supported by WebSphere Application Server and Virtual Member Manager. Also, unless otherwise indicated, any statement about “users” also applies to groups in the user repository. Users and groups are also called “entities”.

Starting with version 6.1, WebSphere Application Server leverages a component called Virtual Member Manager (VMM) to manage information about users, user profiles, and user groups. As a result, user management in WebSphere Portal 6.1 is based on this component and no longer uses WebSphere Member Manager (WMM).

VMM offers a specific Service Provider Interface (SPI), *com.ibm.wsspi.wim.Repository*, to connect VMM (and therefore both WebSphere Application Server and WebSphere Portal) with user repositories that VMM does not support out of the box. This SPI is the replacement for the corresponding interface *com.ibm.websphere.wmm.adapter.MemberRepository* that addressed this use case in WMM.

Despite the fact that the interface addresses the same use case, the *Repository* interface in WebSphere Portal 6.1 is much different than the old WebSphere Member Manager *MemberRepository* interface.

The new interface does not support backward compatibility with the old interface, meaning that existing *MemberRepository* implementations under WebSphere Portal V5.x/6.0.x must be ported to the new interface definition. We explain the differences and give detailed examples in this document.

Just as it was in WMM, all the out-of-the-box-supported user stores (various LDAP servers, a database repository, and the VMM file repository) are accessed through IBM-supplied implementations of the same *Repository* interface that customers can implement for their own unique user repository. This means that this code path has been thoroughly tested.

Note that an implementation of the *Repository* interface is called a “repository adapter”.

2 WebSphere Portal's use of VMM

The APIs exposed by VMM provide an abstract, object-oriented view of users and groups to WebSphere Portal and all portlets that are installed on WebSphere Portal. Details of the individual repository-specific implementations and data layouts remain transparent to code leveraging these APIs.

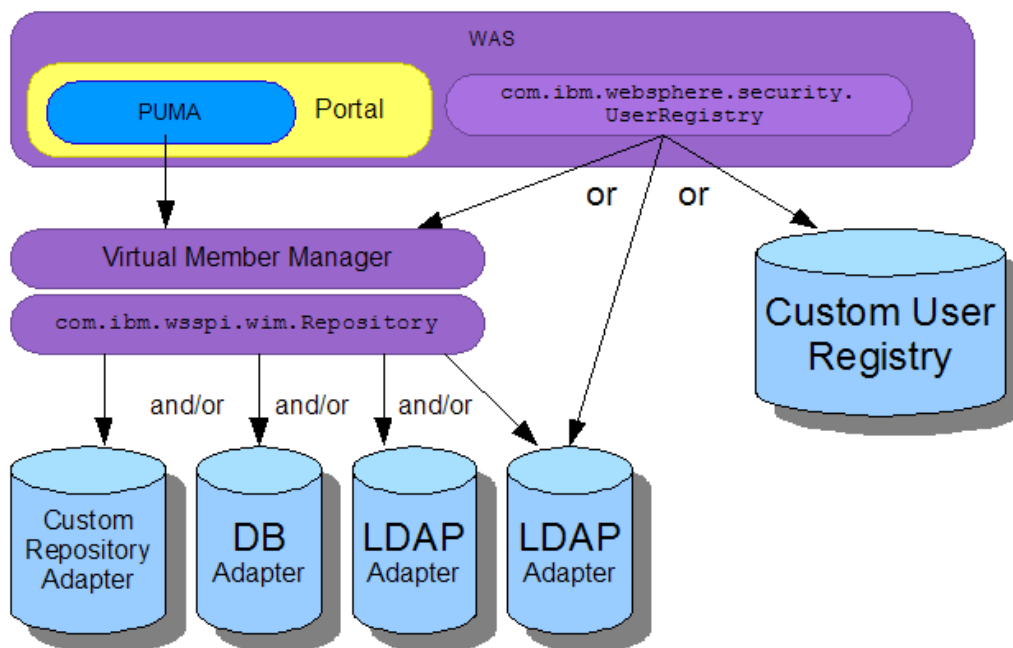
WebSphere Portal v6.1 still uses the public version of Portal User Management Architecture (PUMA) API code on top of VMM. This implementation detail of PUMA is not important for the process of implementing a repository adapter under VMM, but it

will be important for accessing user profile information in custom-written portlets or other WebSphere Portal code.

As the portal and portlets use PUMA to do searches and to get and set attributes on the User object, PUMA passes these requests to the corresponding methods of VMM—which in turn passes them to the corresponding repository adapter.

Figure 1 shows a software stack in which PUMA is always calling the VMM component for user and group information and management. Authentication information is built by WebSphere Application Server Security infrastructure by calling the User Registry for an explicit log-in, or in case of single sign-on scenarios, via implicit log-in.

Figure 1. Software stack for user management



The VMM component can aggregate a set of adapters (the standard File, LDAP, and DB adapters, and custom repository adapters as described in this paper), allowing multiple repositories to be configured.

The User Registry defined for the WebSphere Application Server context can be configured for one of the following three implementations:

- **Virtual Member Manager Federated Repositories.** In this implementation, all operations of the User Registry are dispatched to the VMM component also. When defining a federated repository, you can specify whether VMM only dispatches the code calls to the different repositories, or uses a database to store mappings of external identifiers (used by the repositories) and internal identifiers (used by VMM).

- **Standalone LDAP adapter.** This allows WebSphere Application Server to integrate with a single LDAP directly. If you choose this option, you still must configure VMM appropriately for the WebSphere Portal-related part.
- **Custom User Registry.** This is a custom implementation of the *UserRegistry* interface and is also independent of the VMM part. In scenarios in which both WebSphere Application Server applications and WebSphere Portal must be integrated with custom security backends, the combination of Custom User Registry and Custom Repository Adapter is usually used (described in more detail below).

3 VMM Overview

VMM (as accessed through PUMA) is the abstract interface that WebSphere Portal v6.1 uses to access user and group information. This includes the user accounts that tell WebSphere Portal that the user exists, the enclosing user groups of which the given user might be a member, and the individual attributes of that user. Attributes include information about the user such as user name, job title, or location.

3.1 Types of information managed by VMM

In WebSphere Portal v6.1, VMM manages three types of information about users:

- Authentication information
- The user's memberships in user groups
- The user profile

WebSphere Portal also keeps other information on a per-user basis outside of the VMM component (and therefore not as part of "the user profile" in our definition). For example, the customizations that a user makes to a portlet are stored in portlet instance data in the WebSphere Portal database.

This information is not actually part of the VMM-managed user profile, even though this portlet instance data is in some sense "keyed" by the identity of the user (the user DN, or some other unique identifier for that user). This also includes access control information, which is stored within a separate authorization component in WebSphere Portal.

3.1.1 Authentication information

The actual authentication is not enforced by WebSphere Portal itself, but by WebSphere Application Server, which in turn may further dispatch this enforcement to a separate authentication proxy server, depending on the security configuration of WebSphere Portal.

The authentication process itself typically involves calling the User Registry to verify the user credentials prior to building the corresponding security context. Furthermore, the

application server will call the user registry during enforcement of J2EE authorization constraints to retrieve group membership information.

VMM provides an implementation of the *com.ibm.websphere.security.UserRegistry* interface—the Federated repositories. If WebSphere Application Server is configured to use this implementation, then all the user repositories configured for VMM are automatically available for the log-in.

This means that you do not have to develop a custom implementation of the *UserRegistry* interface if you have a custom user repository; that is, implementing *com.ibm.wsspi.wim.Repository* is sufficient even for log-in. If, however, the Application Server is configured to use a different implementation of *com.ibm.websphere.security.UserRegistry*, you must ensure that VMM is configured to use the same user repository as this implementation does.

When a person logs in to WebSphere Portal, their user account is defined within the user data store that the User Registry implementation is using. For example, if federated repositories is used and VMM is configured towards an LDAP, then an LDAP user object (typically, an *inetOrgPerson* object instance) represents the user account. Such user objects usually have a password that protects that account from unauthorized access.

3.1.2 Group membership

The same user data store that holds the account information is usually the source of group membership information. In LDAP, for example, this takes the form of directory objects that are typically of object class *groupOfNames* or *groupOfUniqueNames*, with membership records pointing to the distinguished name (DN) of the user objects.

Note that the previous LDAP example is how VMM assumes by default that group membership works in the underlying repository in which the members are listed as attributes of the group, rather than the groups being listed as an attribute of the members.

Many directories also support listing the groups of which the user is a member as an attribute of the user object (in Active Directory, for example, this is the *memberOf* attribute).

You can configure VMM to use this attribute when asked by WebSphere Portal for the groups of which a user is a member, rather than doing an LDAP search for objects of the group object class that have the user DN as a member record. This yields performance improvements for such searches.

VMM will still use the member lists of groups when asked to enumerate "all the members of a group." The used data store is responsible for keeping the *memberOf* attribute in sync with the group member list, so that all groups in which the user is listed as a member show up on the attribute, and only groups in which the user is listed as a member show up on the attribute.

VMM also supports cross-repository group membership; that is, a group can have members who are stored in a different repository. It's up to each repository whether it supports groups that have members who are not stored within the same repository. On the other hand, it's up to the configuration to make the group lookup for entities access other repositories. See section 3.4.1.2 for information about the configuration of cross-repository groups.

3.1.3 User profile

The user profile is the set of named and typed attributes representing the individual properties (attributes) of individual users. For example, the user profile might include an attribute named *mailingAddress* of type String, a String *e-mailAddress*, an Integer *serialNumber*, and the *timestamp* of when the user last logged in.

All users in WebSphere Portal and VMM share the same user profile schema. That is, all users have the same set of attribute names as part of their profile, but will often have different values, or no value at all, for those attributes. The profile represents a template for a user, and each user profile is an instance of that template, with at least some of the attributes filled in.

In VMM, the user profile is set by a list of predefined attributes that can be extended with additional attributes defined in a file called *wimxmlextension.xml*. PUMA offers methods to retrieve the list of all defined attributes for users and for groups.

Passwords used for authentication are not part of the user profile, since those are considered authentication information. The distinction between what is and what is not defined as part of the user profile is not really important. The password can be set through interfaces in VMM.

The WebSphere Portal user self-registration and self-care portlets, and the Manage Users and Groups administration portlets, call VMM APIs to set the user's password, if the customer wants to use these WebSphere Portal facilities to do user provisioning.

The passwords are not stored in the WebSphere Portal configuration database but stored with the user objects in the underlying repository. However, it is not possible to read the password from VMM as the password attribute is configured as “sensitive.”

3.1.4 Property Extension repository

The Property Extension repository—formerly known as the Lookaside repository—is used to store entity attributes that cannot be (or that you do not want to be) stored in the main user data store, for example, the LDAP directory. VMM manages this Property Extension repository in its own database.

You can easily define and add new attributes to the user profile and map them to this Property Extension repository. The attributes that are defined in the Property Extension repository are available to all users, regardless of the repository in which they are stored.

3.2 VMM concepts

Each entity stored under VMM is of a dedicated type. Entity types that are used by WebSphere Portal are *PersonAccount* (for users), *Group* (for groups) and *OrgContainer* (for organizations and organizational units). Each entity has a profile describing its characteristics within the system and differentiating one entity from another. After an entity is created, its entity type cannot be changed.

For example, a *Group* cannot be changed to a *PersonAccount*. Each entity is a node in the entity tree (or in one of the entity trees, if there is more than one root node). VMM provides the following features:

- A common mechanism to access entity profiles that are made of attributes, regardless of where and how the data of a member profile is stored. These attributes can be multi-valued and/or composite.
- A set of services to act on and manage entities such as create, read, update, remove, and search entities in user repositories. These services also support managing groups, including assigning members to and removing members from groups, and querying group membership.
- A hierarchical structuring of entities.
- Management of a Property Extension repository within the VMM database to extend and augment any primary user data store.
- A programmatic way to define new attributes in a user data store, if supported by the user data store and the Repository Adapter that connects VMM to that user data store, and if allowed by the customer's access policies to that data store.

3.2.1 VMM functions

VMM supports various entity-related tasks, most of which are passed through to the Repository SPI to be performed by the implementation of that interface.

Entity management tasks:

- Create entities within a hierarchical structure
- Retrieve the profile of an entity using a unique identifier
- Search for entities based on attributes
- Update entities
- Remove entities

Currently not used by WebSphere Portal:

- Move entities within the hierarchical structure.
- Retrieve specific entities, including associations such as the ancestors and descendants of an entity in the hierarchical structure.

Profile repository management tasks:

- List attribute definitions
- Define new attributes (adding new attribute definitions that were not previously "part of the profile") in a profile repository programmatically.
 - Note that the IBM-supplied repository adapters that use LDAP as their data store do not propagate the new attribute definitions to the LDAP server. This means that adding new attribute definitions is only useful for attributes that are already defined in LDAP, but not yet in VMM.
 - Your repository adapter might or might not choose to support adding new attributes programmatically.
 - WebSphere Portal uses this functionality only during the configuration, not during the runtime. Additionally, the WebSphere Portal server must be restarted to make PUMA aware of the recently added attributes.
- Define new attributes in a the Property extension repository, if available

Group management tasks:

- Assign members to groups
- Remove members from groups
- Determine if an entity belongs to a group
- List all members within a group
- List all groups to which an entity belongs

3.2.2 DataObjects

A `com.ibm.sdo.DataObject` is a Java object wrapping some XML and, since all VMM's methods take a `DataObject` as input and returns a `DataObject` as result, the interesting question is: What do these `DataObjects`—the XML they contain—look like?

Chapter 4 below examines each method (and each use case), providing examples for an input `DataObject` and for the resulting output `DataObject`. The examples do not cover all possible shapes supported by VMM, just those currently used by WebSphere Portal code.

3.2.3 Virtual Member Manager unique identifiers

Every entity that is managed by VMM requires a unique identifier, which allows an entity profile to be safely retrieved, preventing ambiguity. VMM provides two types of unique identifiers.

The first, `uniqueName`, is convenient for identification and display purposes. The `uniqueName` is very much like a distinguished name (DN) in LDAP in that it is unique (at least within a certain scope; in this case, within the VMM instance name space, which might at some point encompass multiple physical user repositories) but can be changed, as long as the new value is still unique. (NOTE: The practice of changing DNs is not recommended because many applications treat them as static).

After an entity has been deleted from VMM, a new entity can be created and can reuse the `uniqueName` of the deleted entity. An example of a `uniqueName` is `uid=alice,o=my`

Organization. Note that VMM expects at least a name/value pair (such as name=value) as the uniqueName for an entity.

The uniqueId, on the other hand, is unique, static, and never reused. That is, after a uniqueId for an entity is created, the value of that uniqueId will not change; and even if the entity is deleted, a new entity cannot reuse the value of the uniqueId of the deleted entity.

Therefore, the uniqueName uniquely identifies an **entity** at a single point in time, while the uniqueId, due to its characteristic of never being reused, uniquely identifies a **member** over time.

User repositories such as an LDAP also have the concept of unique identifiers. For instance, LDAP has a DN and a special attribute that serves as extId, such as *ibm-entryuuid*. VMM entities store these values as externalName and externalId. Usually, VMM is configured such that the uniqueName and the user repository's DN match, but it's possible for the repository configuration and/or implementation to define a mapping.

For the uniqueId and externalId it depends on the setup of VMM. Usually, VMM uses the externalId provided by the repository adapter as uniqueId; however, if VMM is configured to use a federation database, then VMM generates its own uniqueIds and stores the mapping of uniqueId to externalId in the database.

Quickly summarized, VMM is responsible for the uniqueId-to-externalId mapping, whereas the repository adapter is responsible for the externalName-to-uniqueName mapping.

When planning a custom VMM adapter implementation, you should carefully consider what your implementation expects and returns as values for the uniqueName, uniqueId, externalName, and externalId attributes within the data objects.

Changing the implementation of this detail may cause inconsistencies and data loss in the WebSphere Portal database. Note that you can partially fix these by running the cleanupusers task (see the [Sample XML configuration](#) topic in the WebSphere Portal 6.1 information center.).

3.2.4 Attributes in VMM

VMM comes with a predefined set of attributes for users and groups. PumaProfile's methods *getDefinedUserAttributeNames()* and *getDefinedGroupAttributeNames()* can be used to retrieve this set at runtime. Additionally, you can execute the *ConfigEngine.bat wp-query-attribute-config* task in directory `wp_profile_root/ConfigEngine`.

If you want to define additional attributes, use the *ConfigEngine.bat wp-add-property* task. The additional attribute definitions are stored in *wimxmlextension.xml*.

For further information, refer to the "[Adapting the attribute configuration](#)" section in the WebSphere Portal 6.1 Information Center .

Member-reference attributes

In addition to the more standard attribute types like Integer or String, there is a special attribute type called IdentifierType, which has a uniqueName as value and thus references another entity. For example, the attribute manager is of IdentifierType and has the uniqueName of the entity's manager as value.

3.3 Mapping user profile attributes to the user repository

If the names of an attribute in the repository and in VMM config do not match, then it's up to the repository adapter to implement an attribute mapping. For example, the attribute *ibm-primaryEmail* is defined in VMM by default and is used by WebSphere Portal to store and retrieve the users' email addresses.

If this attribute is not available in your repository but the email addresses are stored in the attribute *mail*, then your repository adapter could translate a request for *ibm-primaryEmail* into a request for *mail* and return the value of *mail* as the value for *ibm-primaryEmail*.

3.4 Configuration files for VMM

There are two configuration files for VMM:

- <wp_profile_root>/config/cells/<cellName>/wim/config/wimconfig.xml
- <wp_profile_root>/config/cells/<cellName>/wim/model/wimxmlextension.xml.

The *.xsd files contained in those directories provide a good clue of the possible content of the files. The most basic configuration options are discussed in the next sections.

3.4.1 Main configuration file: wimconfig.xml

The *wimconfig.xml* file is used to manage entity types, repositories, and realms.

3.4.1.1 Configuring a repository

For each repository, *wimconfig.xml* contains a <config:repositories> element. This element has an attribute adapterClassName that is used to specify which Java class implements the *com.ibm.wsspi.wim.Repository* interface for that repository. The attribute id assigns a unique name to this repository that is used to identify the repository in the future.

As child elements, there is at least one <config:baseEntries name="o=Default Organization" nameInRepository="o=myRoot"/> element that specifies which DN suffix is contained in that repository and should be made available to VMM. The nameInRepository attribute is optional and can be used to define a DN mapping.

In the above example, the DNs in the repository that end with o=myRoot are accessible to VMM; however, VMM adapter will replace this suffix with o=Default Organization when exposing the entities to WebSphere Portal. If WebSphere Portal asks for a user uid=bob,o=DefaultOrganization, then the repository adapter will convert that DN to

uid=bob,o=myRoot and return the entity referenced by this DN—if it exists. However, the DN returned to WebSphere Portal will again be uid=bob,o=Default Organization.

For a custom repository adapter, there can be any number of `<config:CustomProperties name="aName" value="aValue"/>` elements, and these name/value pairs are passed to the custom adapter during initialization.

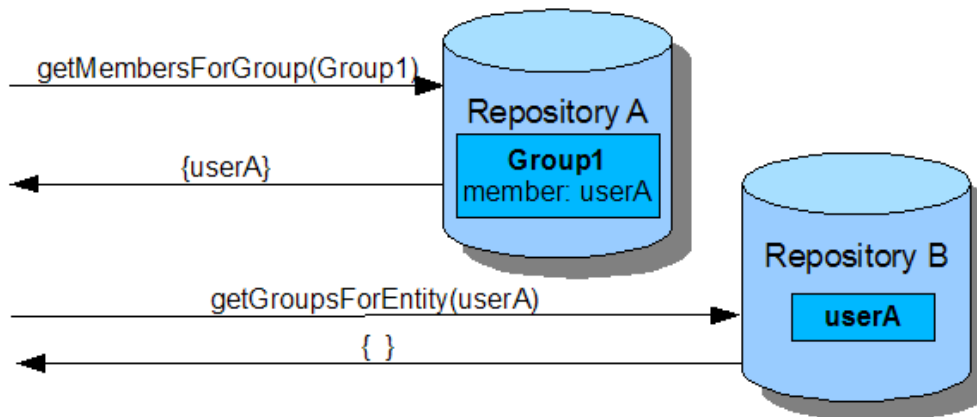
3.4.1.2 Configuring cross-repository group lookup

By default, groups for a given entity are looked up only in the same repository in which the entity is stored. The optional parameter *repositoriesForGroups* causes the specified repository to also be used for group lookup, making it possible to have cross-repository groups, if the corresponding repository adapter supports groups that have members stored in another repository:

```
<config:repositoriesForGroups>aReposId</config:repositoriesForGroups>
```

NOTE: If repository A has groups with members from repository B, but repository B is not configured to have repository A for group lookup, then group membership will not be handled consistently (see figure 2).

Figure 2. Cross-repository group lookup



3.4.1.3 Configuring a realm

The `config:realmConfiguration` element within the `wimconfig.xml` controls the available realms and which is the default realm. Each realm is made up of one or more `participatingBaseEntries`, all of which must occur as a base entry in one of the repositories.

Note that VMM is a bit strict here; it dictates that the very same base entry is configured in any of the repositories, so it is not enough for a repository have a base entry `dc=com` for a realm base entry `dc=yourco,dc=com`.

3.4.1.4 Configuring an entity type

For each entityType, for example, PersonAccount or Group, *wimconfig.xml* contains a supportedEntityTypes element that specifies the default parent DN. This is used when a new entity is created; for instance, when user1 is created, it will have the uniqueName uid=user1,o=defaultWIMFileBasedRealm:

```
<config:supportedEntityTypes defaultParent="o=defaultWIMFileBasedRealm"
name="PersonAccount">
  <config:rdnProperties>uid</config:rdnProperties>
</config:supportedEntityTypes>
```

It is also possible to configure a default parent per realm by adding a <config:defaultParents entityTypeName="PersonAccount" parentUniqueName="dc=com"/> element to the corresponding realm.

3.4.2 Additional custom attributes: wimxmlextension.xml

VMM's out-of-the-box attribute schema can be extended via *wimxmlextension.xml*. This file can contain any number of propertySchema elements that define additional attributes. The attribute definition is always scoped to one or more entityType (see listing 1).

Listing 1. Attribute definition

```
<sdo:datagraph xmlns:sdo="commonj.sdo"
  xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:schema>
    <wim:propertySchema
      nsURI="http://www.ibm.com/websphere/wim" dataType="String"
      multiValued="false" propertyName="mailserver">
      <wim:applicableEntityTypeNames>PersonAccount</wim:applicableEntityTypeNames>
    </wim:propertySchema>
    <wim:propertySchema
      nsURI="http://www.ibm.com/websphere/wim" dataType="String"
      multiValued="false" propertyName="ibm-primaryEmail">
      <wim:applicableEntityTypeNames>Group</wim:applicableEntityTypeNames>
    </wim:propertySchema>
  </wim:schema>
</sdo:datagraph>
```

Note that it is not possible to change an attribute configuration for a predefined attribute with an entry in this file. You can only add new attribute definitions, including an attribute definition for an attribute that is predefined for a different entityType only (as for *ibm-primaryEmail* in the example above).

4 Implementing a custom repository

Now let's discuss how to implement a custom repository.

4.1 Comparing WMM and VMM

Though the functionality is basically the same, VMM's interface for a (custom) repository—*com.ibm.wsspi.wim.Repository*—is much different than the WMM interface *com.ibm.websphere.wmm.adapter.MemberRepository*.

The main difference is that WMM has a dedicated method for each use case, whereas VMM has a small set of more general methods, in which the data passed in as an argument describes not only the subject of the method but also the action.

Table 1 shows how WMM's methods are mapped to VMM's methods. You can see it is an *n*:1 mapping, in which one VMM method addresses several use cases.

Table 1. Mapping WMM's methods to VMM's methods

Methods in WMM	Methods in VMM
createMember	create
createAttributeDefinition createLookasideAttributeDefinition	createSchema
assignmembersToGroup assignMemberToGroup renameMember unassignMemberFromGroup unassignMembersFromGroup updateMember	update
getAncestorIdentifiers getExternalDn getGroupIdentifiersForMember getGroupMemberIdentifiers getGroupMembers getGroupsForMember getMember getMemberIdentifier getMembers getMemberRecursively getDescendantIdentifiers getPersonByAccountName(2) getMemberRecursively isMemberInGroup	get
getAttributeDefinition getAttributeDefinitions getAttributeDataTypes	getSchema

getLookasideAttributeDataTypes	
removeMember	delete
search searchAgain	search
checkPassword mapCertificate	login
moveMember getConfigurationData search(GQL)	-

4.2 Methods of the Repository interface

Since each VMM Repository's method takes a *commonj.sdo.DataObject* as argument, the Repository interface looks quite simple:

```

public abstract void initialize(DataObject dataobject)
    throws WIMException;
public abstract DataObject create(DataObject dataobject)
    throws WIMException;
public abstract DataObject get(DataObject dataobject)
    throws WIMException;
public abstract DataObject delete(DataObject dataobject)
    throws WIMException;
public abstract DataObject update(DataObject dataobject)
    throws WIMException;
public abstract DataObject search(DataObject dataobject)
    throws WIMException;
public abstract DataObject login(DataObject dataobject)
    throws WIMException;
public abstract DataObject createSchema(DataObject dataobject)
    throws WIMException;
public abstract DataObject getSchema(DataObject dataobject)
    throws WIMException;

```

4.2.1 Methods for retrieving entities

The get method offers a variety of possibilities to retrieve entities, the most important of which are:

- get a list of entities by specifying a list of unique identifiers (uniqueName or uniqueId)
- get the (nested) members of a given group, referenced by a unique identifier
- get the (nested) groups for a given entity, referenced by a unique identifier

If a given entity cannot be found, the implementation should throw a *com.ibm.websphere.wim.exception.EntityNotFoundException*. For the WebSphere Portal runtime to work, all three operations must be implemented by a VMM adapter.

4.2.1.1 Getting a list of entities identified by uniqueName or uniqueId

The get method can be used to get one or more entities from VMM by submitting the uniqueNames or uniqueIds of the entities.

In the sample in listing 2 below, two entities are queried, and the repository should include the value of the *properties uid*, *ibm-primaryEmail*, and *manager* in the answer since these attributes are explicitly requested by the PropertyControl element. If the PropertyControl element contains the special value *, then all available attributes should be returned.

Listing 2. Sample get DataObject in which two entities are queried

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>trustEntityType</wim:key>
      <wim:value>>true</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@37e237e2
        (uid: [76e9faf4-5dac-4667-9e13-c81969078cc7])</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@381e381e
    </wim:value>
    </wim:contexts>
    <wim:entities>
      <wim:identifier uniqueName="uid=wpsadmin2,o=Default Organization"/>
    </wim:entities>
    <wim:entities>
      <wim:identifier uniqueName="uid=wpsbind,o=Default Organization"/>
    </wim:entities>
    <wim:controls xsi:type="wim:PropertyControl">
      <wim:properties>ibm-primaryEmail</wim:properties>
      <wim:properties>uid</wim:properties>
      <wim:properties>manager</wim:properties>
    </wim:controls>
  </wim:Root>
</sdo:datagraph>
```

The resulting DataObject could look like that shown in listing 3.

Listing 3. Resulting DataObject

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
```

```

    <wim:identifier externalName="uid=wpsadmin2,o=Default Organization"
repositoryId="fileRepos"
    externalId="uid=wpsadmin2,o=Default Organization"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
    <wim:uid>wpsadmin2</wim:uid>
    <wim:ibm-primaryEmail>wpsadmin2@test.com</wim:ibm-primaryEmail>
  </wim:entities>
  <wim:entities xsi:type="wim:PersonAccount">
    <wim:identifier externalId="uid=wpsbind,o=Default Organization"
externalId="uid=wpsbind,o=Default Organization"
    repositoryId="fileRepos"
uniqueName="uid=wpsbind,o=Default Organization"/>
    <wim:uid>wpsbind</wim:uid>
    <wim:ibm-primaryEmail>wpsadmin@test.com</wim:ibm-primaryEmail>
  </wim:entities>
</wim:Root>
</sdo:datagraph>

```

Note that the identifier contains the externalName, the externalId, the uniqueName and the repositoryId, which is the unique name of the repository (see section 3.4.1.1).

4.2.1.2 Get the members of a given group

If the DataObject passed as argument to a get method contains a GroupMemberControl element, the repository is asked to return the members of the requested group. The level property specifies whether only direct members are to be returned (level=1) or the members of subgroups should be resolved recursively (level=0).

For the core WebSphere Portal usage (log-in and access control), it is not necessary to implement this functionality. However, both the administrative portlets for user management and PUMA API expose it, which is why we recommend implementing this functionality in the custom adapter, too (see listing 4).

Listing 4. Sample get DataObject that requests the members of wpsadmins group

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>trustEntityType</wim:key>
      <wim:value>true</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@511c511c
(uid: [elfa03b1-56d6-4ald-a7e1-768ed1ea26fa])
      </wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@51585158
      </wim:value>
    </wim:contexts>
    <wim:entities>
      <wim:identifier uniqueId="cn=wpsadmins,o=Default Organization"
uniqueName="cn=wpsadmins,o=Default Organization"/>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

```

    <wim:controls xsi:type="wim:GroupMemberControl" level="1">
      <wim:properties>uid</wim:properties>
    </wim:controls>
  </wim:Root>
</sdo:datagraph>

```

The meaning of the properties element is that the entities included in the resulting DataObject should include the value of uid (see listing 5).

Listing 5. Resulting DataObject

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:Group">
      <wim:identifier externalId="cn=wpsadmins,o=Default Organization"
externalName="cn=wpsadmins,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="cn=wpsadmins,o=Default Organization"/>
      <wim:members xsi:type="wim:PersonAccount">
        <wim:identifier externalId="uid=wpsadmin2,o=Default Organization"
externalName="uid=wpsadmin2,o=Default Organization"
          repositoryId="fileRepos"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
        <wim:uid>wpsadmin2</wim:uid>
      </wim:members>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

4.2.1.3 Get the groups for a given entity

If the DataObject passed as argument to a get method contains a GroupMembershipControl element, the repository is asked to return the groups of which the requested entity is a member (see listing 6). The level property specifies whether only direct groups are to be returned (level=1) or the groups of groups should be resolved recursively (level=0).

Listing 6. Sample get DataObject that requests the groups of user wpsadmins2

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>trustEntityType</wim:key>
      <wim:value>true</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@7c887c88
(uID: [ala616eb-4ca3-4e93-9153-924406a162be])</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@7cc47cc4</wim:value>
    </wim:contexts>
  </wim:Root>
</sdo:datagraph>

```

```

    </wim:contexts>
    <wim:entities>
      <wim:identifier uniqueId="uid=wpsadmin2,o=Default Organization"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
    </wim:entities>
    <wim:controls xsi:type="wim:GroupMembershipControl" level="1">
      <wim:properties>cn</wim:properties>
    </wim:controls>
  </wim:Root>
</sdo:datagraph>

```

The meaning of the properties element is that the groups included in the resulting DataObject should include the value of *cn* (see listing 7).

Listing 7. Resulting DataObject

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="uid=wpsadmin2,o=Default Organization"
externalName="uid=wpsadmin2,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
      <wim:groups>
        <wim:identifier externalId="cn=wpsadmins,o=Default Organization"
externalName="cn=wpsadmins,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="cn=wpsadmins,o=Default Organization"/>
        <wim:cn>wpsadmins</wim:cn>
      </wim:groups>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

4.2.2 Creating new entities

The create method is used for creating a new entity, for example, a new user (PersonAccount) or group (Group) in the repository. You do not need to implement this method if you don't want to support user and group creation from WebSphere Portal. A possible input DataObject looks like that shown in listing 8.

Listing 8. Sample create DataObject that creates a new user

```

<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@23362336
(uid: [ 205ad427-1660-4b2f-88ed-4850b6ee623e ])</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@237
22372</wim:value>
    </wim:contexts>
  </wim:Root>
</sdo:datagraph>

```

```

<wim:entities xsi:type="wim:PersonAccount">
  <wim:identifier uniqueName="uid=newUserID,o=defaultWIMFileBasedRealm"/>
  <wim:parent>
    <wim:identifier uniqueName="o=defaultWIMFileBasedRealm"/>
  </wim:parent>
  <wim:password>KioqKg==</wim:password>
  <wim:uid>newUserID</wim:uid>
  <wim:cn>newUserFirstName newUserLastName</wim:cn>
  <wim:sn>newUserLastName</wim:sn>
  <wim:preferredLanguage>en</wim:preferredLanguage>
  <wim:ibm-primaryEmail>newUserMail@ibm.com</wim:ibm-primaryEmail>
  <wim:givenName>newUserFirstName</wim:givenName>
</wim:entities>
</wim:Root>
</sdo:datagraph>

```

The interesting part of the XML is the entities element, which characterizes the entity that is to be created. In the example, the repository is requested to create a new User (PersonAccount) with a specified uniqueName. The uniqueName of the parent node is specified in the parent element. Additionally, the entities element contains a set of properties, given as name-value pairs.

If the adapter can handle the request and create such a User object, it returns a DataObject like that shown in listing 9.

Listing 9. Resulting DataObject confirming successful creation of new user

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="c336771f-cf7b-4e64-a4d3-c77cca78890f"
externalName="uid=newUserID,o=defaultWIMFileBasedRealm"
uniqueId="c336771f-cf7b-4e64-a4d3-c77cca78890f"
      <wim:parent>
        <wim:identifier uniqueName="uid=newUserID,o=defaultWIMFileBasedRealm"/>
      </wim:parent>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

The resulting entities element contains the same uniqueName as given in the input. The externalName is the identifier used to store the user in the repository. It's up to the implementation of the repository whether the uniqueName or a different identifier (for example, uid=newUserID,o=yourco) is used as external name.

It's the other way round with the externalId and uniqueId: The implementation chooses an externalId to identify the entity and returns it in the DataObject.

4.2.3 Deleting entities

The delete method is used to delete an entity. You don't need to implement this method if it doesn't need to be supported to delete users or groups from WebSphere Portal (see listings 10 and 11).

Listing 10. Incoming DataObject for deleting the user toDelete

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@2c242c24
(uID: [8e34c44d-71d1-4108-81a5-1f9f41deb19d])</wim:value>
      </wim:contexts>
      <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@2c602c60
      </wim:value>
      </wim:contexts>
      <wim:entities>
      <wim:identifier externalId="1d9966a6-ef61-4ab7-b86a-dd83b01c6de7"
externalName="uid=toDelete,o=defaultWIMFileBasedRealm"
      repositoryId="InternalFileRepository"
uniqueId="1d9966a6-ef61-4ab7-b86a-dd83b01c6de7"
      uniqueName="uid=toDelete,o=defaultWIMFileBasedRealm"/>
      </wim:entities>
      <wim:controls xsi:type="wim:DeleteControl" deleteDescendants="true"/>
    </wim:Root>
  </sdo:datagraph>
```

Listing 11. Result DataObject

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="1d9966a6-ef61-4ab7-b86a-dd83b01c6de7"
externalName="uid=toDelete,o=defaultWIMFileBasedRealm"
      repositoryId="InternalFileRepository"
uniqueId="1d9966a6-ef61-4ab7-b86a-dd83b01c6de7"
      uniqueName="uid=toDelete,o=defaultWIMFileBasedRealm"/>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>
```

4.2.4 Getting meta data

During startup of WebSphere Application Server, VMM asks each repository for the list of available properties per entity type, using the getSchema method. The request—for example, for getting the properties that are available for entity type Group—looks like that shown in listing 12.

Listing 12. Sample DataObject for Group property definitions request

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:controls xsi:type="wim:PropertyDefinitionControl"
repositoryId="InternalFileRepository" entityType="Group"/>
  </wim:Root>
</sdo:datagraph>
```

```
</wim:Root>
</sdo:datagraph>
```

The response DataObject should include the intersection of those properties that VMM has defined for the entityType and those properties the repository can handle for that entityType. Note that the overall VMM schema can be accessed at any time by use of the *com.ibm.wspi.wim.SchemaHelper* class.

The result DataObject might look like listing 13.

Listing 13. Result DataObject

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:sdo="commonj.sdo"
  xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:schema>
      <wim:propertySchema propertyName="identifier"/>
      <wim:propertySchema propertyName="viewIdentifiers"/>
      <wim:propertySchema propertyName="parent"/>
      <wim:propertySchema propertyName="children"/>
      <wim:propertySchema propertyName="groups"/>
      <wim:propertySchema propertyName="createTimestamp"/>
      <wim:propertySchema propertyName="modifyTimestamp"/>
      <wim:propertySchema propertyName="entitlementInfo"/>
      <wim:propertySchema propertyName="partyRoles"/>
      <wim:propertySchema propertyName="cn"/>
      <wim:propertySchema propertyName="members"/>
      <wim:propertySchema propertyName="displayName"/>
      <wim:propertySchema propertyName="description"/>
      <wim:propertySchema propertyName="businessCategory"/>
      <wim:propertySchema propertyName="seeAlso"/>
    </wim:schema>
  </wim:Root>
</sdo:datagraph>
```

4.2.5 Logging in

If WebSphere Application Server is configured to use VMM as its User Registry, then a log-in will lead to a call of the login method. This means that, if the User Registry is configured for stand-alone LDAP or a Custom User Registry, this method is not called and doesn't need to be implemented by the VMM adapter.

The incoming DataObject holds the username (principalName), password, and the searchbase where the user should be looked up (see listing 14). The username can be either the short name or the full uniqueName, and the password can be accessed with the following piece of code:

```
byte[] password =
dataObject.getDataObject(GenericHelper.DO_FIRST_ENTITY)
.getBytes(Service.PROP_PASSWORD);
```

Listing 14. Sample DataObject for logging in wpsadmin2

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>realm</wim:key>
      <wim:value>defaultWIMFileBasedRealm</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>allowOperationIfReposDown</wim:key>
      <wim:value>>false</wim:value>
    </wim:contexts>
    <wim:entities xsi:type="wim:LoginAccount">
      <wim:principalName>wpsadmin2</wim:principalName>
      <wim:password>KioqKg==</wim:password>
    </wim:entities>
    <wim:controls xsi:type="wim:LoginControl">
      <wim:properties>principalName</wim:properties>
      <wim:searchBases>o=Default Organization</wim:searchBases>
    </wim:controls>
  </wim:Root>
</sdo:datagraph>

```

When there's a successful authentication (user is found and password matches), the repository adapter should return a `DataObject` that looks like listing 15.

Listing 15. Resulting `DataObject` indicating a successful authentication

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="uid=wpsadmin2,o=Default Organization"
externalName="uid=wpsadmin2,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
      <wim:principalName>uid=wpsadmin2,o=Default
Organization</wim:principalName>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

If the user cannot be found, then the repository adapter should return an empty `DataObject` - `SDOHelper.createRootDataObject()`. If there is an exception, for example, the user is found but the password does not match, then an exception must be thrown.

4.2.6 Searching for entities

For the search method, the `SearchControl` element of the incoming `DataObject` contains the search expression (an XPath expression), the searchBase (affecting the uniqueName, not the externalName), and a list of those properties that should be included in the response (see listing 16).

Listing 16. Sample input `DataObject` that searches users with `uid=wps*`

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:controls xsi:type="wim:SearchControl" countLimit="0"
expression="@xsi:type='PersonAccount' and uid=&quot;wps*&quot;"
timeLimit="600000">
      <wim:properties>cn</wim:properties>
      <wim:properties>uid</wim:properties>
      <wim:searchBases>o=Default Organization</wim:searchBases>
    </wim:controls>
  </wim:Root>
</sdo:datagraph>

```

Implementation of this method is also required for the WebSphere Portal runtime. A VMM adapter implementation should support at least those search expressions that are handled by the sample adapter implementation provided in the Downloads section of this paper.

The above sample input DataObject could result in the output DataObject in listing 17.

Listing 17. Output DataObject

```

<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="uid=wpsbind,o=Default Organization"
externalName="uid=wpsbind,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="uid=wpsbind,o=Default Organization"/>
      <wim:principalName>uid=wpsbind,o=Default
Organization</wim:principalName>
      <wim:uid>wpsbind</wim:uid>
      <wim:cn>Wps Admin</wim:cn>
    </wim:entities>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="uid=wpsadmin2,o=Default Organization"
externalName="uid=wpsadmin2,o=Default Organization"
      repositoryId="fileRepos"
uniqueName="uid=wpsadmin2,o=Default Organization"/>
      <wim:principalName>uid=wpsadmin2,o=Default
Organization</wim:principalName>
      <wim:uid>wpsadmin2</wim:uid>
      <wim:cn>Wps Admin</wim:cn>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>

```

4.2.7 Updating attributes and membership information

When the attributes of an entity are changed, the input DataObject will contain the new state of the entity along with a change summary indicating what's been changed. Custom VMM adapters don't need to implement the update method, if write operations to users or groups from WebSphere Portal are not permitted.

Listing 18 shows an example of adding a value for the *ibm-primaryEmail* attribute.

Listing 18. Adding a value for *ibm-primaryEmail* attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sdo="commonj.sdo"
  xmlns:sdo_1="http://www.eclipse.org/emf/2003/SDO"
xmlns:wim="http://www.ibm.com/websphere/wim">
  <changeSummary xmlns="">
    <objectChanges key="#//@RootObject/@root/@entities.0">
      <value xsi:type="sdo_1:EChangeSummarySetting"
featureName="ibmPrimaryEmail"
      set="false"/>
    </objectChanges>
  </changeSummary>
  <wim:Root>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>
      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@48364836
(uID: [b3ba0cba-c7c2-4e83-946f-38aa9048627f])</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>
com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@48724872
      </wim:value>
    </wim:contexts>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="bfeld222-b5d6-4ba8-972e-25c17acda338"
externalName="uid=wpsadmin,o=defaultWIMFileBasedRealm"
      repositoryId="InternalFileRepository"
uniqueId="bfeld222-b5d6-4ba8-972e-25c17acda338"
      uniqueName="uid=wpsadmin,o=defaultWIMFileBasedRealm"/>
      <wim:password>KioqKg==</wim:password>
      <wim:uid>wpsadmin</wim:uid>
      <wim:cn>wpsadmin</wim:cn>
      <wim:sn>wpsadmin</wim:sn>
      <wim:ibm-primaryEmail>wpsadmin@test.com</wim:ibm-primaryEmail>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>
```

Note that the entities element contains the new value for *ibm-primaryEmail*, whereas the changeSummary contains the old state, namely, that no value was set (see listing 19).

Listing 19. Result DataObject

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:PersonAccount">
      <wim:identifier externalId="bfeld222-b5d6-4ba8-972e-25c17acda338"
externalName="uid=wpsadmin,o=defaultWIMFileBasedRealm"
      repositoryId="InternalFileRepository" uniqueId="bfeld222-b5d6-4ba8-
972e-25c17acda338"
      uniqueName="uid=wpsadmin,o=defaultWIMFileBasedRealm"/>
    </wim:entities>
  </wim:Root>
</sdo:datagraph>
```

The code in listing 20 can be used to remove a member from a group.

Listing 20. Remove a member from a group

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:contexts>
      <wim:key>PLUGIN_UID_CONTEXT</wim:key>

      <wim:value>com.ibm.ws.wim.pluginmanager.context.impl.UIDContextImpl@f940f94
      (uID: [55e7302d-9dba-4c0b-b29d-886349175d5c])</wim:value>
    </wim:contexts>
    <wim:contexts>
      <wim:key>PLUGIN_EXECUTION_CONTEXT</wim:key>
      <wim:value>
      com.ibm.ws.wim.pluginmanager.context.impl.ExecutionContextImpl@fd00fd0
      </wim:value>
    </wim:contexts>
    <wim:entities xsi:type="wim:Group">
      <wim:identifier externalId="c6da3de9-8cba-4519-aae8-81dcf1ee3994"
      externalName="cn=wpsadmins,o=defaultWIMFileBasedRealm"
      repositoryId="InternalFileRepository"
      uniqueId="c6da3de9-8cba-4519-aae8-81dcf1ee3994"
      uniqueName="cn=wpsadmins,o=defaultWIMFileBasedRealm"/>
      <wim:members>
        <wim:identifier externalId="bfe1d222-b5d6-4ba8-972e-25c17acda338"
        repositoryId="InternalFileRepository"
        uniqueId="bfe1d222-b5d6-4ba8-972e-25c17acda338"
        uniqueName="uid=wpsadmin,o=defaultWIMFileBasedRealm"/>
      </wim:members>
    </wim:entities>
    <wim:controls xsi:type="wim:GroupMemberControl" modifyMode="3"/>
  </wim:Root>
</sdo:datagraph>
```

Note that modifyMode controls the type of modification to be performed, with the following possible values:

- 1 = add specified entities as members
(SchemaConstants.VALUE_MODIFY_MODE_ASSIGN)
- 2 = remove all members and set given entities as new members
(SchemaConstants.VALUE_MODIFY_MODE_REPLACE)
- 3 = remove specified members
(SchemaConstants.VALUE_MODIFY_MODE_UNASSIGN)

The result DataObject looks like listing 21.

Listing 21. Result DataObject

```
<?xml version="1.0" encoding="UTF-8"?>
<sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sdo="commonj.sdo" xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:Root>
    <wim:entities xsi:type="wim:Group">
```

```

    <wim:identifier externalId="c6da3de9-8cba-4519-aae8-81dcf1ee3994"
externalName="cn=wpsadmins,o=defaultWIMFileBasedRealm"
    repositoryId="InternalFileRepository" uniqueId="c6da3de9-8cba-4519-
aae8-81dcf1ee3994"
    uniqueName="cn=wpsadmins,o=defaultWIMFileBasedRealm"/>
  </wim:entities>
</wim:Root>
</sdo:datagraph>

```

4.2.8 Initializing the repository

The initialize(DataObject) method is called once during VMM initialization at server startup. When VMM is initialized, it retrieves the configuration settings from *wimconfig.xml* and provides part of the information to each configured repository.

In fact, the entire content of the repositories element that configures the repository is passed as argument to the initialize method (see listing 22).

Listing 22. Sample input DataGraph for initialize method

```

<config:repositories
adapterClassName="vmm.sample.adapter.FileRepositorySampleAdapter"
id="fileRepos" isExtIdUnique="true" supportExternalName="false"
supportPaging="false" supportSorting="false" supportTransactions="false">

  <config:CustomProperties name="readOnly" value="true"/>
  <config:CustomProperties name="logFile"
value="/shared/wp610/wp_profile/logs/WebSphere_Portal/adapter.log"/>
  <config:CustomProperties name="enableTrace" value="true"/>
  <config:CustomProperties name="userRegistryFile"
value="/shared/wp610/wp_profile/config/cells/wpsbvt/UserRepository.xml"/>
  <config:CustomProperties
name="userMap.uniqueMemberIdentifier" value="uniqueId"/>
  <config:CustomProperties name="userMap.uid" value="userId"/>
  <config:CustomProperties name="userMap.givenName" value="firstName"/>
  <config:CustomProperties name="userMap.sn" value="lastName"/>
  <config:CustomProperties name="userMap.ibm-primaryEmail" value="email"/>
  <config:CustomProperties name="userMap.preferredLanguage"
value="preferredLanguage"/>
  <config:CustomProperties name="userMap.phone" value="phone"/>
  <config:CustomProperties name="userMap.cn" value="cn"/>
  <config:CustomProperties name="userMap.dn" value="uniqueId"/>
  <config:CustomProperties name="groupMap.cn" value="groupId"/>
  <config:CustomProperties name="groupMap.dn" value="uniqueId"/>
  <config:CustomProperties name="groupMap.uniqueMemberIdentifier"
value="uniqueId"/>
  <config:CustomProperties
name="groupMap.description" value="description"/>

  <config:baseEntries name="o=Default Organization"
nameInRepository="o=Default Organization"/>
</config:repositories>

```

In this way, the initialize method has access to all configuration preferences that affect the repository, as shown in the snippet example in listing 23.

Listing 23. Code to access the adapter's custom properties in initialize method

```

List<DataObject> customProps =
dataObject.getList(ConfigConstants.CONFIG_DO_CUSTOM_PROPERTIES);

for (DataObject customProp: customProps)
{
String propName = prop.getString(ConfigConstants.CONFIG_PROP_NAME);
//=> "readOnly"
String propValue = prop.getString(ConfigConstants.CONFIG_PROP_VALUE);
//=> "true"
}

```

See Section 5.1.1 for more information on setting up `wimconfig.xml` to load and use your custom adapter.

4.2.9 Creating a new attribute definition

The `createSchema` method is called if a schema change is performed that does not affect the general VMM config, only the particular repository. This is the case if a new attribute definition is to be added for that particular repository. VMM then updates the `wimxmlextension.xml` and calls the `createSchema` method on the affected repository afterward.

It's now up to the repository whether any action must be taken to update any persistent configuration maintained outside of VMM (for example, execution of some SQL to add the new attribute definition to a database).

5 Setting up a custom repository

NOTE: Make sure to execute all the steps below before configuring a database repository or property extension repository.

5.1 Using the custom repository within a federated repository

First let's review how to use the custom repository within a federated repository.

5.1.1 Add custom repository as additional federated repository

When you have finished implementing your custom repository and want to start using / testing it, follow these instructions to enable the new repository:

1. Deploy the `.jar` file that contains your implementation by placing it in a directory that belongs to WebSphere Application Server's classpath, for example, `<install_root>/AppServer/lib`.
2. Add your custom repository to VMM's federated repository configuration:
 - a. Edit the `wkplc.properties` file, located in the `wp_profile_root\ConfigEngine\properties` directory.

- b. Enter the appropriate values for the following properties (sample values for the sample file repository are given here):

```
federated.cur.id=myCustomRepository
federated.cur.adapterClassName
=vmm.sample.adapter.file.RepositoryAdapterFile
federated.cur.baseDN=o=Default Organization
federated.cur.isExtIdUnique=true
federated.cur.supportExternalName=false
federated.cur.supportPaging=false
federated.cur.supportSorting=false
federated.cur.supportTransactions=false
```

- c. Save the file and make sure that WebSphere Portal Server is running.

- d. Run the `ConfigEngine.bat wp-create-cur` task in the `wp_profile_root/ConfigEngine` directory.

3. You have now added the custom repository to the VMM configuration. However, depending on the implementation of your repository, you might need to add additional properties. To do so, use the following properties in *wkplc.properties*:

```
cur.id=myCustomRepository
cur.name=myCustomProperty
cur.value=myValue
```

and execute the `ConfigEngine.bat wp-create-cur-custom-property` task in the `wp_profile_root/ConfigEngine/` directory.

For the sample registry, use the values shown in listing 24 (adopting the paths to your environment):

Listing 24. Values for configuring the sample adapter as VMM repository

```
cur.name=readOnly cur.value=false
cur.name=logFile cur.value=c:/ibm/wp_profile/logs/WebSphere Portal/adapter.log
cur.name=enableTrace cur.value=true
cur.name=userRegistryFile
cur.value=c:/ibm/wp_profile/config/cells/wpsbvt/UserRepository.xml
cur.name=userMap.uniqueMemberIdentifier cur.value=uniqueId
cur.name=userMap.uid cur.value=userId
cur.name=userMap.givenName cur.value=firstName
cur.name=userMap.sn cur.value=lastName
cur.name=userMap.ibm-primaryEmail cur.value=email
cur.name=userMap.preferredLanguage cur.value=preferredLanguage
cur.name=userMap.phone cur.value=phone
cur.name=userMap.cn cur.value=cn
cur.name=userMap.dn cur.value=uniqueId
cur.name=groupMap.cn cur.value=groupId
cur.name=groupMap.dn cur.value=uniqueId
cur.name=groupMap.uniqueMemberIdentifier cur.value=uniqueId
cur.name=groupMap.description cur.value=description
```

For the sample registry, place the *UserRepository.xml* file at the location specified by property `cur.name=userRegistryFile` (see above), and then restart the WebSphere Portal server.

5.1.2 Assign administrator role to user in custom repository (optional)

To change the WebSphere Portal administrator and administrative group, use the following command:

```
ConfigEngine.bat wp-change-portal-admin-user -DnewAdminPw=wpsadmin2  
-DnewAdminId="uid=wpsadmin2,o=Default Organization" -  
DnewAdminGroupId="cn=domain expert,o=Default Organization"
```

in directory [wp_profile_root](#)/ConfigEngine/.

To change the WebSphere Application Server administrative user, run the following command:

```
ConfigEngine.bat wp-change-was-admin-user -DnewAdminPw=wpsadmin2 -  
DnewAdminId="uid=wpsadmin2,o=Default Organization"
```

in directory [wp_profile_root](#)/ConfigEngine/.

Note that both changes require a server restart to take effect.

5.1.3 Delete other repositories from VMM configuration (optional)

After you have assigned all administrative roles to users and groups hosted by your custom repository, you can remove the other repositories, such as the default-file-based repository:

1. Edit the *wkplc.properties* file,
located in the [wp_profile_root](#)\ConfigEngine\properties directory.
2. Enter the appropriate values for the following properties (sample values to delete the default-file-based repository are given here):

```
federated.delete.baseentry=o=defaultWIMFileBasedRealm  
federated.delete.id=InternalFileRepository
```

3. Save the file and make sure that the WebSphere Portal server is running.
4. Run the `ConfigEngine.bat wp-delete-repository` task in the [wp_profile_root](#)/ConfigEngine directory.

5.2 Using the custom repository as a standalone registry

If you want to use your custom repository as a standalone registry, you must implement the *com.ibm.websphere.security.UserRegistry* interface, which is used by WebSphere Application Server during authentication.

All subsequent requests to the user repository coming from WebSphere Portal are still executed against VMM, meaning that you must implement the *com.ibm.wsspi.wim.Repository* interface, too.

Of course, both implementations might use a common class to handle the requests. You can bundle both implementations in one .jar file or keep them separate, but in any case:

1. Place them in a directory that belongs to WebSphere Application Server's classpath, for example, <install_root>/AppServer/lib.
2. Edit the *wkplc.properties* file, located in the `wp_profile_root\ConfigEngine\properties` directory.
3. Enter the appropriate values for the properties in listing 25 (sample values for the sample file repository and registry are given here):

Listing 25. Values for using the sample registry implementation as UserRegistry

```
standalone.cur.id=myCustomRepository
standalone.cur.baseDN=o=Default Organization

standalone.cur.realm=anyRealmName
standalone.cur.delimiter=/

standalone.cur.adapterClassName=vmm.sample.adapter.file.RepositoryAdapterFile
standalone.cur.WasAdapterClassName=vmm.sample.adapter.file.CustomUserRegistryFile

standalone.cur.propertyName= userRegistryFile
standalone.cur.propertyValue
=c:/ibm/wp_profile/config/cells/wpsbvt/UserRepository.xml

standalone.cur.primaryAdminId=uid=wpsadmin2,o=Default Organization
standalone.cur.primaryAdminPassword=wpsadmin2
standalone.cur.primaryPortalAdminId=uid=wpsadmin2,o=Default Organization
standalone.cur.primaryPortalAdminPassword=wpsadmin
standalone.cur.primaryPortalAdminGroup=cn=testadmins,o=Default Organization
standalone.cur.personAccountParent=o=Default Organization
standalone.cur.groupParent=o=Default Organization
standalone.cur.personAccountRdnProperties=uid
standalone.cur.groupRdnProperties=cnstandalone.cur.isExtIdUnique=true
standalone.cur.supportExternalName=false
standalone.cur.supportPaging=false
standalone.cur.supportSorting=false
standalone.cur.supportTransactions=false
```

4. Run the `ConfigEngine.bat wp-modify-cur-security` task in directory <wp_profile_root>/ConfigEngine.

5. To add additional properties to the VMM configuration of your repository, follow step 3 in Section 5.1.1.
6. For the sample adapter, place the *UserRepository.xml* file at the location specified with property `cur.name=userRegistryFile` (see above).
7. Stop the WebSphere Portal server.
8. To configure additional properties for your custom registry, open the *security.xml* file (located in directory `<wp_profile_root>/config/cells/<cellName>`) and add the properties shown in listing 26 (for the sample file registry).

Listing 26. Sample file registry properties

```
<userRegistries xmi:type="security:CustomUserRegistry"
xmi:id="CustomUserRegistry_1" useRegistryServerId="false"
primaryAdminId="uid=wpsadmin2,O=defaultWIMFileBasedRealm"
customRegistryClassName="vmm.sample.adapter.file.CustomUserRegistryFile">
  <properties xmi:id="Property_1218188583671"
name="User_Registry_File"
value="c:/ibm/wp_profile/config/cells/wpsbvt/UserRepository.xml"
required="false"/>
</userRegistries>
```

9. Restart WebSphere Portal.

6 Tips and tricks

How to use VMM code when implementing a custom adapter:

Constants

The interface *com.ibm.websphere.wim.SchemaConstants* contains many constants that are useful when parsing and constructing the DataObjects.

Utility classes

Here are some examples of VMM classes that you might find useful:

- *com.ibm.websphere.wim.util.SDOHelper*: class for creating new DataObjects
- *com.ibm.websphere.wim.util.SDOUtils*: class for serializing a DataObject (e.g., for tracing)
- *com.ibm.wsspi.wim.GenericHelper*: some general helper Methods
- *com.ibm.wsspi.wim.SchemaHelper*: gives you access to the current VMM schema (e.g., the defined properties)

For code snippets, refer to the sample adapter in the Downloads section.

7 Verifying your implementation

To verify your custom VMM adapter implementation, you can code unit tests against VMM in “offline” mode, if you have a local WebSphere Application Server 6.1 version on your development machine.

Along with the sample code, a sample test project is provided in the Downloads section (RepositoryTest.zip), which you can use as a starting point for developing your own unit tests. Follow these instructions to set up a test project:

1. Import the RepositoryTest project as an existing project to your Eclipse workspace.
2. Add the following .jar files as external libraries to the project:
 - com.ibm.ws.runtime_6.1.0.jar from <App Server>\plugins
 - org.eclipse.emf.*.jar (all jar files starting with that prefix) from <App Server>\plugins
 - com.ibm.ws.emf_2.1.0.jar from <App Server>\plugins
 - j2ee.jar from <App Server>\lib
 - junit library
 - your custom adapter .jar file
3. Configure your custom repository adapter in the VMM configuration files of your local WebSphere Application Server as described in Section 5.
4. In the *wimconfig.xml* file, additionally set the attribute `isSecurity="false"` in the `config:authorization` tag.
5. For the Eclipse project, define a run configuration and set the following VMM arguments:
 - `-Dlocal.cell=<WAS cell name>`
 - `-Duser.install.root=<WAS profile directory>`

8 The File Repository sample adapter

The read-only custom adapter in the Downloads section is based on an XML file as user storage. Here we provide a brief overview of the implementation.

8.1 Architecture

For any implementation of *com.ibm.wsspi.wim.Repository* you must perform the following steps for (almost) each method:

1. Parse the incoming DataObject to determine the exact request.
2. Execute the request (for example, search for users) in the repository backend.

3. Create and return a new `DataObject` as result.

NOTE: Steps 1 & 3 are not specific to the underlying user repository backend. In other words, it doesn't matter whether the user repository backend that is to be connected to VMM is an LDAP, a database, a file, etc.; Steps 1 & 3 are always identical.

The sample implementation that accompanies this paper takes advantage of this by clearly separating Step 1 & 3 from Step 2:

- *Identifier*, *RepositoryEntity*, and *AttributeDefinition* are data classes / interfaces that represent the content of the backend repository.
- The interface *RepositoryAdapterReadOnly* defines methods for read access to the repository backend (used in step 2). These methods are loosely coupled to VMM code as they don't use `DataObject` class; instead, they use *Identifier*, *RepositoryEntity*, and *AttributeDefinition* as parameters and return types.
- The abstract class *GenericAdapterReadOnly* implements both the *RepositoryAdapterReadOnly* interface and the *com.ibm.wsspi.wim.Repository* interface. The class is abstract as it does not provide an implementation for the *RepositoryAdapterReadOnly* methods.

Each of `Repository`'s methods first parses the `DataObject` (Step 1), then calls the appropriate method(s) in *RepositoryAdapterReadOnly*, and then finally translates the result into a `DataObject` (Step 3).

- *RepositoryAdapterFile* extends *GenericAdapterReadOnly* and provides an implementation for the *RepositoryAdapterReadOnly* methods (Step 2). Any other custom repository implementation can reuse *GenericAdapterReadOnly* (for Steps 1 & 3) by extending it and just re-implementing the *RepositoryAdapterReadOnly* interface (for the backend-specific Step 2).

IMPORTANT: All the interfaces and classes in the Downloads are samples only; they are not meant to be used as production code or SPI.

9 Conclusion

This white paper started with an overview of WebSphere Portal's use of VMM and continued with an introduction into VMM's core concepts and its configuration. We examined the differences between WMM's SPI and VMM's SPI and gave a detailed description of the VMM SPI interface, including code samples.

Also included are instructions for using a custom repository adapter, some hints for the implementation, and a description of unit-testing a custom adapter without running a WebSphere Application server. Finally, we described the sample file adapter in the Downloads section.

10 Resources

- developerWorks® article, “IBM WebSphere Developer Technical Journal: Expand your user registry options with a federated repository in WebSphere Application Server V6.1”:
http://www.ibm.com/developerworks/websphere/techjournal/0701_ilechko/0701_ilechko.html
- WebSphere Application Server information center, JavaDoc for com.ibm.wsspi.wim:
<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.javadoc.doc/vmm/com.ibm/wsspi/wim/package-summary.html>
- WebSphere Portal information center, “Planning your user registry” topic:
http://publib.boulder.ibm.com/infocenter/wpdoc/v6r1m0/topic/com.ibm.wp.ent.doc/plan/plan_ureg.html
- developerWorks WebSphere product page:
<http://www.ibm.com/developerworks/websphere/>

11 About the authors

Stefan Schmitt is the WebSphere Portal Security Architect and team lead. He joined IBM in 2000 and has worked in the WebSphere Portal development team since the 1.0 release. He is the IBM WebSphere Portal Vulnerability contact, responsible for the architecture and design of the user management functions in WebSphere Portal. Prior to joining IBM, Stefan studied Information Technology at the University of Cooperative Education, Stuttgart.

Andreas Zehnpfenning received his Diploma of Computer Science from the University of Stuttgart. After completing his diploma thesis on speech synthesis at IBM in 2005, he joined the WebSphere Portal team, working on various security-related topics. He is currently responsible for the VMM integration into WebSphere Portal.

Jan-Paul Buchwald is an IT Specialist at the IBM Development Laboratory in Boeblingen, Germany. He received a Diploma of Computer Science at the Albert Einstein University of Ulm. After several years of experience in developing WebSphere Portal, Jan-Paul joined the WebSphere Portal Lab Services team in 2008.

Trademarks

developerWorks, IBM, and WebSphere are registered trademarks or trademarks of the International Business Machines Corporation in the United States and/or other countries.

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.