HP OpenView Select Identity

Connector Developer Guide

Software Version: 3.0

July 2004

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Portions Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved.

Select Identity uses software from the Apache Jakarta Project including:

- Commons-beanutils.
- Commons-collections.
- Commons-logging.
- Commons-digester.
- Commons-httpclient.

- Element Construction Set (ecs).
- Jakarta-poi.
- Jakarta-regexp.
- Logging Services (log4j).

Additional third party software used by Select Identity includes:

- JasperReports developed by SourceForge.
- iText (for JasperReports) developed by SourceForge.
- BeanShell.
- Xalan from the Apache XML Project.
- Xerces from the Apache XML Project.
- Java API for XML Processing from the Apache XML Project.
- SOAP developed by the Apache Software Foundation.
- JavaMail from SUN Reference Implementation.
- Java Secure Socket Extension (JSSE) from SUN Reference Implementation.
- Java Cryptography Extension (JCE) from SUN Reference Implementation.
- JavaBeans Activation Framework (JAF) from SUN Reference Implementation.
- OpenSPML Toolkit from OpenSPML.org.
- JGraph developed by JGraph.
- Hibernate from Hibernate.org.

Trademark Notices

HP OpenView Select Identity is a trademark of Hewlett-Packard Development Company, L.P.

Microsoft, Windows, the Windows logo, and SQL Server are trademarks or registered trademarks of Microsoft Corporation.

Sun™ workstation, Solaris Operating Environment™ software, SPARCstation™ 20 system, Java technology, and Sun RPC are registered trademarks or trademarks of Sun Microsystems, Inc. JavaScript is a trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

This product includes the Sun Java Runtime. This product includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

IBM, DB2 Universal Database, DB2, WebSphere, and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

This product includes software provided by the World Wide Web Consortium. This software includes xml-apis. Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institute National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

AMD and the AMD logo are trademarks of Advanced Micro Devices, Inc.

BEA and WebLogic are registered trademarks of BEA Systems, Inc.

VeriSign is a registered trademark of VeriSign, Inc. Copyright © 2001 VeriSign, Inc. All rights reserved.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Support

Please visit the HP OpenView web site at:

http://openview.hp.com/

There you will find contact information and details about the products, services, and support that HP OpenView offers.

You can go directly to the support web site at:

http://support.openview.hp.com/

The support web site includes:

- Downloadable documentation
- Troubleshooting information
- Patches and updates
- Problem reporting
- Training information
- Support program information

contents

1

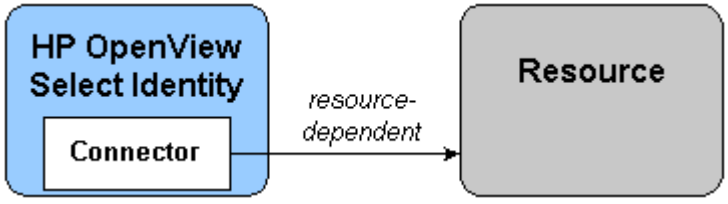
Introduction to Connectors

HP OpenView Select Identity enables you to connect to enterprise applications and resources to configure and manage users, groups, and entitlements in those systems. The component that enables Select Identity to access a resource is called a **connector**.

Overview of Select Identity Connectors

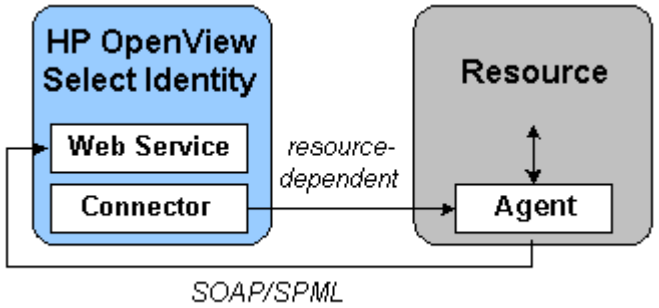
Select Identity supports two types of connectors:

- A one-way connector initiates communication with a resource. If a resource is supported through the use of a one-way connector, user data must be updated in Select Identity and the changes are sent to the resource, to synchronize the resource with Select Identity. The following diagram illustrates the flow of data:



The connector resides on the Select Identity server and sends requests to the resource. The resource defines the protocol that must be used by the connector to issue the request. To create a one-way connector, you must create the connector and install it on the Select Identity server.

- A two-way connector comprises the connector that resides on the Select Identity server and an agent that resides on the resource. This enables the resource to issue a request to Select Identity, as well as Select Identity issuing a request to the resource. Thus, a two-way connector enables data to flow in two directions, as illustrated in the following diagram, and changes to users can occur on either system.



The connector must issue a request according to the resource's specifications. When the resource issues a request to Select Identity's web service, it must use the SOAP protocol, sending an SPML payload.

J2EE Connector Architecture (JCA)

Creating a Select Identity connector entails building a J2EE Connector architecture (JCA) resource adapter. JCA provides a Java solution to the problem of connectivity between application servers and enterprise information systems (EISs). This architecture is based on technologies that are defined and standardized as part of the Java 2 Platform, Enterprise Edition (J2EE).

For a general overview of JCA, refer to the following page:

http://java.sun.com/j2ee/white/connector.html

You can download the specification from the following page:

http://java.sun.com/j2ee/connector/download.html

Note that Select Identity implements only the Connector Management portion of the JCA specification.

You must have an understanding of the Java Developer Kit (JDK), version 1.4, and you should be familiar with the JCA, version 1.0. In addition, Select Identity has provided a Connector API that is used in conjunction with JCA to create connectors.

For information about the J2EE APIs, including those for connectors, refer to **http://java.sun.com/j2ee/1.4/docs/api/index.html**.

Development Phases

This section outlines the steps that are typically involved in the development of a connector. It is strongly recommended that you take the time to address each phase and plan for the connector's development carefully.

Requirements Phase

Ensure that the resource supports a mechanism for user provisioning by external clients, in a secure and reliable manner. You must have an understanding of the underlying resource, including knowledge of the resource's tools and administration API. You may also need to obtain an administrative account that has privileges to provision.

Collect requirements for development, as follows:

1. Determine the requirements based on the resource system.
   - What identity information will be provisioned? What users or other objects?
   - What are the entitlements supported by the resource? Typically, resources support groups (groups or users), roles, access control levels (ACLs), privileges, and so on.
   - What are the supported attributes of the identity object based on the schema in the resource?
   - How is the schema retrieved from the resource?

- How is the identity object addressed on the resource? This could be a DN (for LDAP-type of resources), an SSN,a user ID, hierarchical naming, and so on. This will be used as the primary key to address the identity object.
- How does the resource application support connectivity for external systems to provision identity information? This might mean accessing the system through API calls, RMI, JMS, a Web Service, a CLI such as telnet, ssh, and so on.
- If the resource already supports a connector interface, how can you develop the Select Identity connector leveraging the existing connector?
- Does the resource support an SDK or a development toolkit for administration, which might include JAR files or libraries for making calls to access and provision information?
- Are there security requirements to consider? Is SSL or any proprietary encryption/decryption information required between the connector and the resource?
- What are the performance requirements? How many objects can the resource support? How may entitlements? How many users can the connector create, read, update, or delete in a second, minute, or hour?
- What are the scalability requirements? How many connections does it support? Can the same connector support similar resources through configuration support for transactions?
- Does the resource support synchronous or asynchronous connectivity? It is possible that the resource cannot finish provisioning immediately and might finish the job at a later time. How does the connector know when the resource operation is done and how does it handle the response from the resource?
- Is the connector required to maintain state? If so, what is the required schema?

2. Determine access requirements for the resource.
   - What are the addressing parameters such as TCP/IP address, port number, URL,and secure IDs?
   - Is there authentication information (user ID and password)?
   - Are there secure channel parameters?

- Does the connection pass through a proxy server or a firewall? If so, what are the parameters involved?

3. Determine the requirements for error reporting.
   - What errors are supported by the resource?
   - What kind of exceptions are reported to Select Identity?
   - What kind of errors in the resource are reported to Select Identity?
   - What are the recoverable and non-recoverable exceptions?
4. Determine the requirements for reverse synchronization.
   - What changes to identity objects on the resource must be synchronized with Select Identity. For example, if a user's password or address changes on the resource, is there a requirement that Select Identity should be notified about this?
   - How often do changes occur? Are they done in real time or as a batch job at the end of the day?
   - How is information obtained from the resource? The resource might support an audit log of all changes on the resource, or it might support a log of all events that are triggered by someone like an administrator. How is this information retrieved from the resource? Should the connector support a pull model or a push model?
5. Detemine the requirements for child transactions.
   - Is an operation invoked on the resource that might trigger child operations within the resource?
   - How should the connector notify Select Identity of the status of child operations?
   - What status information about child operations should be reported to Select Identity?
   - Is the operation is “atomic” or a “best-effort?”
   - How does the connector determine when the operation is done?
   - Does the resource automatically rollback all previous successful child operations if one child operation fails?
6. Determine requirements for the policies supported by the resource.

- What are the policies for the identity objects? For example, the primary key of the identity object must be obtained from another external system.
- What are the attribute policies? For example, password policy might restrict in the size, content (maximum lenght, minimum length, maximum number of alphabetic characters, minimum number of numeric charactrs, and so on), encryption (one-way or two-way), and so on. What are the limitations on attribute size, masking, and other parameters?

Design Phase

Design the connector you will implement following these guidelines:

1. Provide a high-level design of the approach taken for the provisioning process. Provide the following:
   - Mapping of functionality to be supported by the connector to the functionality supported by the resource.
   - Mapping of the Select Identity schema to the schema (attribute information) supported by the resource. This is also referred to as the forward mapping.
   - The Connector API methods that are supported by the connector implementation.
   - Reverse mapping of the attribute information at the time of reverse synchronization.
   - How the implementation solves the cyclic update problem. For example, a change in object's information triggers an update on the resource, which might in turn trigger a reverse synchronization with Select Identity for the same object, and vice-versa.
   - Use of the JCA framework in the design. Define how the connector makes use of the framework to address some of the requirements.
   - Resource product version. Provide any functionality changes between versions of the resource application.
2. Provide information about how to address the various requirements: synchronous versus asynchronous processing, scalability, performance, security, and so on.

- — Can the connector handle large number of identity objects, such as users?
- — Can it handle large number of entitlements? Is caching, paging, batch loading, or file loading is used by the connector?
- — Can it handle large number of resources?

3 Define whether the connector is agent-based or agent-less.

- — Agent-based requires that an agent is installed on the resource with which the connector implementation interacts. The agent in turn interacts with the resource or the operating system. Reverse synchronization is generally possible with an agent-based solution. On the other hand, an agent-based implementation requires an installation effort and administration on the resource system.
- — An agent-less connector requires complete out-of-box support for all provisioning operations by the resource or through an SDK.
- — Address the advantages and disadvantages for both solutions.

Implementation

Specific information about how to implement the JCA and Connector API methods is provided in Interface, Class, and Method Implementations on page 22. This procedure provides a general overview.

1 Start with a sample application that can provision identity objects and perform entitlement assignment s on the identity objects in the resource.

2 Implement all of the required methods of the Select Identity Connector API to create, read, update, and identity objects, leveraging the sample application.

- — The main interface to implement is TAConnector interface.
- — Implement the connector parameter factory, which creates instances of connection parameter beans.

3 Implement all entitlement association and dissociation methods.

4 Implement all required interfaces in the JCA CCI framework, which enables the connector as a Resource Adapter.

5 If necessary, implement an agent to run on the resource machine.

6. Implement a secure way of communication between the connector and resource, and vice versa. If necessary, use certificates.
7. Implement modules to send SOAP messages containing SPML to the Select Identity Web Service for reverse synchronization (password synchronization and identity object reverse synchronization).
8. If necessary, install the EJB driver for unit testing the connector.
9. If necessary, install the client driver for testing the connector.
10. Use IDEs for the development and ANT for build tools.
11. Use the JDK, J2EE, and third-party libraries for further development.
12. Implement junit test cases.

Integration

Verify the connector's integration with Select Identity as follows:

1. Verify that Select Identity is loading and using the connector as a resource adapter to communicate with the new resource.
2. Create a Service that uses this resource.
3. Provision users in the Service, verifying that they are successfully created in the resource.
4. Associate and disassociate entitlements with users.
5. Verify integration with the Select Identity Web Service for user provisioning through SPML payloads.

Packaging

Detailed information about packaging the connector is described in Building a Connector on page 20. This provides a general overview:

1. Include all libraries required by the connector in a RAR file.
2. Test the client for unit testing.
3. Determine any schema information (ddl, dml) needed by the connector.
4. Obtain all third-party software licenses and their installation procedures.

Documentation

For future maintenance and distribution, compile the following information about the connector:

- Detailed documentation on the requirements and design
- User guides
- Configuration guides
- Functionality mapping document
- Schema (or attribute) mapping document
- Installation guides, for agent-less and agent-based solutions
- Javadoc
- Documentation of encryption/decryption used, port numbers of agent, size of agent foot print, and so on
- Requirements on the system administrator to install the agent on the resource
- Administration documents

2

Implementing a Connector

You must have an understanding of the Java Developer Kit (JDK), version 1.4, and you should be familiar with the JCA, version 1.0. In addition, Select Identity has provided a Connector API that is used in conjunction with JCA to create connectors.

You can download the JCA specification from the following page:

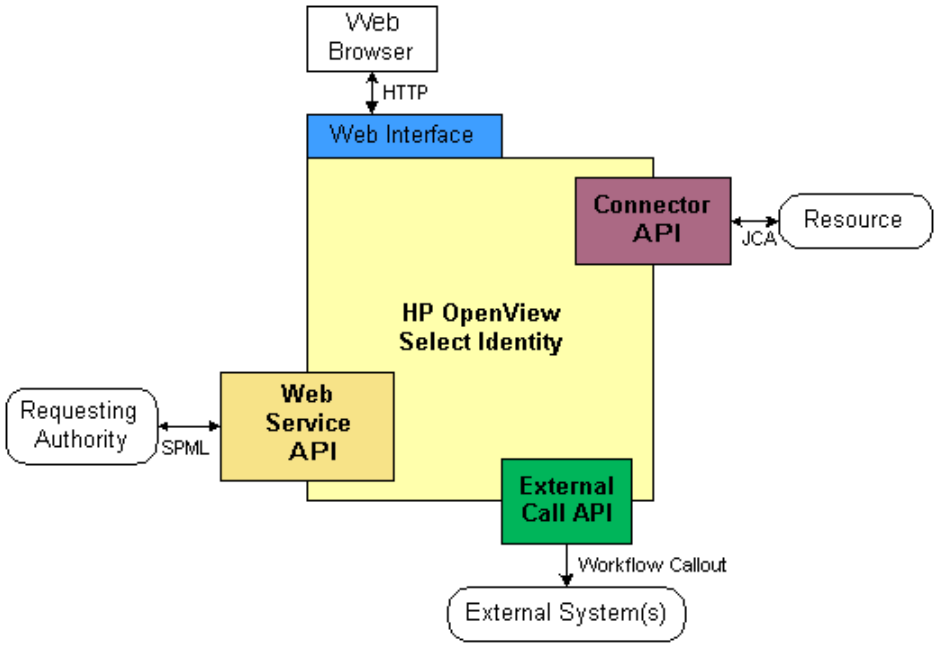
http://java.sun.com/j2ee/connector/download.html

For information about the J2EE APIs, including those for connectors, refer to **http://java.sun.com/j2ee/1.4/docs/api/index.html**.

When implementing a connector using the J2EE Connector APIs and the APIs described here, it is expected that the operations on the connector instances are called within transactions and from multiple threads. Also, the connectors must implement adequate synchronization to prevent data corruption.

Overview of the Select Identity Connector API

The following diagram illustrates the Select Identity API architecture, showing the relationship of the Connector API to Select Identity and the other APIs:



The following classes and interfaces are provided by the Connector API. Online help (Javadoc) is provided for this API on the Select Identity CD, in the `docs/connectors/Javadoc` directory:

- **EntitySupport**

  Defines the actions that can be performed on an entity, which is an object that is managed by Select Identity, such as a user, group, role, or stage.

- **GroupModel**

  Represents an entitlement on a resource.

- **RelationSupport**

  Specifies an association between identity object types, such as between a user and entitlement and vice versa.

- **TAConnector**

  The top-level interface that maps identity information to a resource type.

- **TAConnectorFactory**

  Creates instances of connections handles for resources. The connection handle is an implementation of TAConnector.

- **TAConnectorParamBean**

  Describes a configuration parameter needed by the connector. Examples of such parameters include URLs or configuration parameters like wait time. Select Identity retrieves a list of these beans to create a user interface to obtain values from the user.

- **TAConnectorParameterFactory**

  Obtains connection-specific beans that contain connection parameter values.

- **TAConnectorParamValueBean**

  An abstract class that represents the connection parameter values needed to establish a connection to a resource. It also contains all parameters needed to access the resource for user provisioning.

- **TAFilter**

  Enables you to issue search requests.

- **UserEntitySupport**

  Provides additional methods for setting permissions on UserModel classes. It shows the level of support for user objects in the repository. In addition to supporting create, read, update, and delete tasks, UserEntitySupport specifies whether the password can be reset or changed in the resource.

- **UserModel**

  Enables you to retrieve and set attributes for users; returns a list of users.

Building a Connector

To build a connector, follow this general procedure. Each step below references another section that provides further details. You may also find it useful to refer to LDAP Connector Example on page 42 for an overview of those source files.

1. If you have not done so, review Development Phases on page 10 in preparation for this procedure.
2. Code the Java classes and implement the interfaces that will comprise the connector. This entails implementing JCA and Select Identity Connector APIs. You must also register the parameter factory implementation with JNDI. See Interface, Class, and Method Implementations on page 22 for details.
3. Create a mapping file that maps each attribute on the physical resource to an attribute on the connector. See Creating a Mapping File on page 35 for more information.
4. Build the connector. Select Identity provides all of the base classes you need to build a connector in a file named `clientintf.jar`, which resides on the Select Identity CD. Use the contents of this file to build your connector. See Description of Build Files on page 47 for a listing of build files created to build the LDAP connector.
5. *Two-way connector only*
   Code the file(s) that will comprise the agent. The resource type will dictate the programming language and APIs you use.

   When the agent sends data to the Select Identity server, it must send a Service Provisioning Markup Language (SPML) compliant message. It must also send the data using SOAP. Refer to the User Provisioning API for more information about the SPML-compliant message.
6. Define the deployment descriptor by creating an XML file called `ra.xml`. This file contains deployment specific information; you must specify the interface class names and implementation class names of the connector here. Here is an example of the "resourceadapter" section of the descriptor taken from LDAP connector:

```
<resourceadapter>
  <managedconnectionfactory-class>com.trulogica.truaccess.
   connector.ldap.ldapv3.LDAPManagedConnectionFactory
  </managedconnectionfactory-class>
```

```
  <connectionfactory-interface>com.trulogica.truaccess.
   connector.TAConnectorFactory
  </connectionfactory-interface>
  <connectionfactory-impl-class>com.trulogica.truaccess.
   connector.ldap.ldapv3.LDAPConnectorFactory
  </connectionfactory-impl-class>
  <connection-interface>com.trulogica.truaccess.connector.
   TAConnector
  </connection-interface>
  <connection-impl-class>com.trulogica.truaccess.connector.ldap.
   ldapv3.LDAPConnector
  </connection-impl-class>
  <transaction-support>NoTransaction</transaction-support>
  <reauthentication-support>false</reauthentication-support>
</resourceadapter>
```

Create this XML file according to the JCA specification.

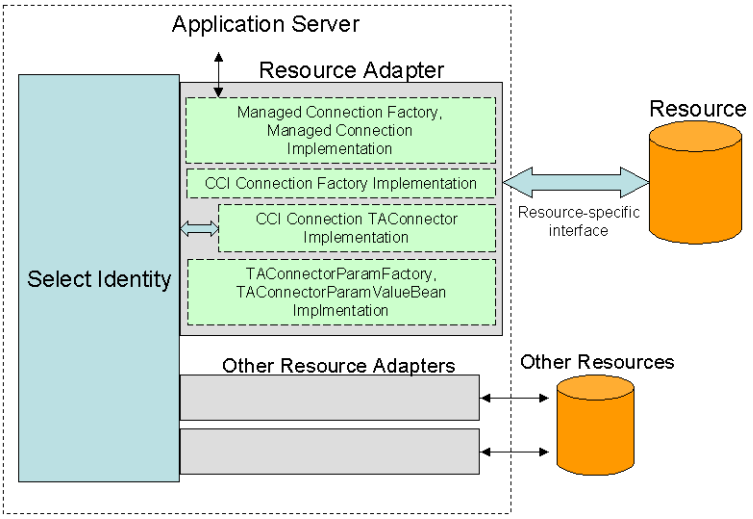
7 Create the WebLogic-specific deployment descriptor by creating a file called `weblogic-ra.xml`. You must register the JNDI name for the connector (`eis/`*`connector`*) here. The following is an example:

```
<weblogic-connection-factory-dd>
  <connection-factory-name>
    LDAPConnectorFactory
  </connection-factory-name>
  <jndi-name>eis/LDAPv3</jndi-name>
  <pool-params>
    <initial-capacity>0</initial-capacity>
  </pool-params>
</weblogic-connection-factory-dd>
```

8 Bundle all other connector class files, library JAR files, and the `ra.xml` file in a RAR file. Use the following format when naming this file: *`connector`*`.rar`.

Interface, Class, and Method Implementations

The following illustrates the connector architecture and the relationship between connector classes, Select Identity, and the resource:



The following provides guidelines that you must follow when coding the connector.

JCA Interfaces

Implement the JCA interfaces; refer to the J2EE specification for details. Connectors must support both local and distributed transaction protocols. If a resource does not support transactions, the adapter must record transactions so that compensating transactions can be applied on rollback. Regarding security, JAAS is used to authenticate inbound communication.

Here are the interfaces you must implement:

- java.resource.cci.Connector extended by TAConnector
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory (implement only one instance of this interface)
- javax.resource.spi.ManagedConnectionMetaData

Select Identity Connector API Interfaces and Classes

Implement the following Select Identity Connector API interfaces and classes. Refer to the API online help (on the Select Identity CD, in the `docs/connector/Javadoc` directory) for more information.

- TAConnector interface — the main interface to implement. Select Identity calls on the methods in this interface to perform provisioning operations. In particular, you *must* implement these methods:

Method	Comment
add(UserModel)	**To implement:** Build all of the needed attributes by taking the values from UserModel and creating a new user on the resource. After a succesfully adding the user, Select Identity requires the implementation to set the key field value by calling UserModel.setUserId(). **Usage:** The add(UserModel) method is called to provision a new user on the resource.
changePassword(UserModel)	**To implement:** Use the new password in UserModel to update the password on the resource. **Usage:** This method is called to change a user's password.
expirePassword(UserModel, Boolean)	**Usage:** This method expires a password for the specified user.

Method	Comment
get(UserModel)	**To implement:** The connector should construct the key field value of the user on the resource and check the existence of user in the resource. If the user is not present on the resource, the connector should throw ObjectNotFoundException. **Usage:** This method is called to verify the existence of user in the resource.
getGroupAttributes()	**To implement:** The implementation can return one attribute to represent the ID of the entitlement. **Usage:** This method must return the schema supported by the resource for group and entitlement provisioning.
getGroups()	**To implement:** Build and return a set of Strings that identify the entitlement. **Usage:** This is called by Select Identity to get a list of all entitlements on the resource. For example, this could be used when your are setting the fixed entitlements for a service, or it could be used when you are adding a user.
getGroups(TAFilter)	**To implement:** Build and return a set of Strings that identify the entitlement. Using TAFilter, you can filter out the list returned. **Usage:** This is called by Select Identity to get a list of all entitlements on the resource. TAFilter is used to filter out the entitlements retrieved from the resouce and is passed in from the filter.

Method	Comment
getUserAttributes()	**To implement:** Return a list of TAConnectorParamBean instances that contain details about each of the attributes supported by the connector. These attributes are the Select Identity resource attributes and not the attributes on the physical resource. The mapping between the Select Identity resource attributes and the physical resource attributes is to be done by the connector. For example, the LDAP connector uses an XML mapping file to map these attributes. In this mapping file, "tafield" is the Select Identity resource attribute and "resfield" is the physical resource attribute. **Usage:** Select Identity calls this method to get the schema supported by the connector for the user object.
isPasswordValid(password)	**To implement:** Check the validity of the password on the resource. **Usage:** This method is called before adding a new user or resetting the password of an existing user.
isSupported(entity1)	**To implement:** Return the support for UserModel and GroupModel. GroupModel is a generic container that represents any type of entitlement on the resource. Examples include user groups, access control levels, privileges, roles, and so on. **Usage:** This method is called on the connector to get the level of support for the object.

Method	Comment
isSupported(entity1, entity2)	**Usage:** This method is called to get the level of support for the association of entitlements to users.
link(UserModel, GroupModel)	**To implement:** Select Identity first makes a call to getGroups(TAFilter) to get a list of all entitlements on the resource. Then, GroupModel passed in references to the entitlements returned by the getGroups method. You can call getGroupId(), which identifies the entitlement and uses it as the key to associate this user with the entitlement on the resource. **Usage:** This method is used to assign entitlements to an existing user. Select Identity calls this method once for each entitlement to be assigned.
remove(UserModel)	**To implement:** Delete the user account on the resource. **Usage:** This method is called to remove a user from a resource.
resetPassword(UserModel)	**To implement:** Use the new password in UserModel to update the password on the resource. **Usage:** This method is called to reset a user's password.

Method	Comment
setStatus(UserModel, int)	**To implement:** Depending on the resource support that is appropriate, the implementation could disable or enable the user on the resource. For example, on a resource that does not support this feature, you can set an attribute to reflect that the user is disabled or enabled. If the connector implementation does not support this operation, throw NotImplementedException. When the connector throws this exception, Select Identity will call back on the connector to add or delete all of the entitlements on the user (using link or unlink methods). This feature is provided to support those resources where the meaning of disable or enable is to remove or add entitlements of the user, respectively. **Usage:** This method is called when user is being disabled or enabled for all Services.

Method	Comment
test()	**To implement:** This method should test the connectivity of the physical resource using the connection parameter values given in the implementation of TAConnectorParamValueBean. This method can also implement the required logic to validate the connection parameters. **Usage:** Select Identity calls this method on the connector when a new resource is deployed or an existing resource is modified. Select Identity expects the connector to verify connectivity with the resource and validate the connection parameters.
unlink(UserModel, GroupModel)	**To implement:** Dissociate the user from the entitlement referred to by the GroupModel.getGroupId(). **Usage:** This is called when a user's Service membership is modified or when user is disabled for a given Service.
update(UserModel)	**To implement:** Update the attributes of the user on the resource. **Usage:** This method is called to update the attributes of a user. Select Identity sends all attributes of the user.

- TAConnectorFactory interface — creates instances of connection handles for the connector. The connection handle is an implementation of the TAConnector interface. For the TAConnectorFactory interface, you must implement the following method:

Method	Comment
getConnection(connParam)	**Usage:** This is called to return the implementation of the TAConnector interface.

- TAConnectorParameterFactory interface — obtains connector-specific beans that hold connection parameter values. In particular, you must implement these methods:

Method	Comment
createParamValueBean()	**Usage:** This is called to create a bean to pass to the parameter values.
getParamBeans()	**Usage:** This is called to return a collection of TAConnectorParamBean classes if the connector needs configuration values from the user.

- TAConnectorParamValueBean class — an abstract class that represetns the connection parameter values needed to establish a connection with the resource. It also holds all parameters needed to access the resource for user provisioning. In particular, you must implement these methods:

Method	Comment
getTAInstallDirectory()	**Usage:** This is called to return the path of the Select Identity installation directory.
setTAInstallDirectory(path)	**Usage:** This is called to set the path of the Select Identity installation directory.
getParamNames	**To implement:** Return all connection parameter names. **Usage:** This method will return the names of the connection parameters that are used to establish a connection with the resource.

Method	Comment
get	**To implement:** Return the value of the connector parameter. **Usage:** This gets the value of the connection parameter.
set	**To implement:** Save the value of the connector parameter. **Usage:** The set method sets the connection parameter value. Select Identity will pass the values provided when the resource was deployed using this method. The bean implementation should store the value for later use.

You can also implement the following Select Identity Connector API interfaces and classes (this is a subset of all interfaces and classes provided by the API), as needed:

- UserModel
- GroupModel
- EntitySupport
- RelationSupport

Finally, implement an authentication mechanism for the connection. For example, you may implement simple username/password authentication using an administrative account that has the necessary authority.

JNDI Registration of the Parameter Factory Implementation

You must register the parameter factory implementation with JNDI. Select Identity will look up the parameter factory when creating instances of TAConnectorParamValueBeans.

The following sample code illustrates how you could register the parameter factory implementation with the JNDI on the application server. Select Identity will reference this factory and use it to create instances of ParamValueBean in which it passes the connection information.

```
private void registerParamFactory(String connectorJndiName)
throws Exception
{
  String lFuncName = "registerParamFactory()";
  LDAPParamFactory paramFactory = new LDAPParamFactory();
  InitialDirContext initCtx = new InitialDirContext();

  // Initialize the factory
  paramFactory.initialize();

  try {
    initCtx.lookup("eis");
  } catch (NameNotFoundException e) {
    initCtx.createSubcontext("eis");
  }

  // Register param factory with JNDI
  // Example: eis/LDAPv3-ParamFactory
  String lPfJndiName = connectorJndiName +
    TAConnectorParameterFactory.JNDI_PARAMFACTORY_SUFFIX;
  try {
    initCtx.lookup(lPfJndiName);
    catch (NameNotFoundException e) {
    initCtx.bind(lPfJndiName, paramFactory);
  } finally {
      initCtx.rebind(lPfJndiName, paramFactory);
    }
  }
```

Mapping File Overview

As described in Building a Connector on page 20, you must create a file that maps the Select Identity fields defined for a user to the fields used by the resource. The connector will reference this mapping file to understand the target field on the resource for each user value.

The LDAP connector provides three mapping files: one for an Active Directory server (`ActiveDir.xml`), one for an iPlanet server (`iPlanet.xml`), and one for ETrust (`CAEtrust.xml`). The files are created in XML, according to SPML standards, and are bundled in a JAR file called `schema.jar`.

General Attribute Information

The following operations can be performed in the mapping file:

- Add a new attribute mapping
- Delete an existing attribute mapping
- Modify attribute mappings

Here is an explanation of the elements in the XML mapping files provided by the LDAP connectors:

- **<Schema>**, **<providerID>**, and **<schemaID>**

  Provides standard elements for header information.

- **<objectClassDefinition>**

  Defines the actions that can be performed on the specified object as defined by that name attribute (in the <properties> element block) and the Select Identity-to-resource field mappings for the object (in the <memberAttributes> block). For example, the object class definition for users defines that users can be created, read, updated, deleted, reset, and expired in LDAP.

  - **<properties>**
    Defines the operations that are supported on the object. This can be used to control the operations that are performed through Select Identity. The following operations can be controlled:

    — Create (CREATE)

    — Read (READ)

    — Update (UPDATE)

    — Delete (DELETE)

    — Enable (ENABLE)

    — Disable (DISABLE)

    — Reset password (RESET_PASSWORD)

    — Expire password (EXPIRE_PASSWORD)

    — Change password (CHANGE_PASSWORD)

The operation is assigned as the name of the <attr> element and access to the operation is assigned to a corresponding <value> element. You can set the values as follows:

- true — the operation is supported by the connector
- false — the operation is not supported by the connector and will throw a permission exception
- bypass — the operation is not supported by the connector but will not throw any exception; the operation is simply bypassed

Here is an example:

```
<objectClassDefinition name="User" description="Active
Directory User>
   <properties>
     <attr name="CREATE">
       <value>true</value>
     </attr>
     <attr name="READ">
       <value>true</value>
     </attr>
```

- **<memberAttributes>**
  Defines the attribute mappings. This element contains <attributeDefinitionReference> elements that describe the mapping for each attribute. Each <attributeDefinitionReference> must be followed by an <attributeDefinition> element that specifies details such as minimum length, maximum length, and so on.

  Each <attributeDefinitionReference> element contains the following attributes:

  - Name — the name of the reference.
  - Required— if this attribute is required in the provisioning (set to true or false).
  - Concero:tafield — the name of the Select Identity resource attribute.
  - Concero:resfield — the name of the physical resource attribute from the resource schema. If the resource does not support an explicit schema (such as UNIX), this can be a tag field that indicates a resource attribute mapping.

— Concero:isKey — An optional attribute that, when set to true, specifies that the connector can use this attribute mapping to locate the object on the resource. To set the attribute mapping as a key, you need to specify this attribute for only one <attributeDefinitionReference> element in the <memberAttributes> element. The connector attribute for which you set isKey does not need to be the same attribute that is defined as the key in Select Identity.

— Concero:init — An optional attribute that identifies that the attribute is initialized with the value of the attribute passed in from Select Identity.

Here is an example:

```
<memberAttributes>
  <attributeDefinitionReference name="User Name"
   required="true" concero:tafield="[User Name]"
   concero:resfield="cn" concero:isKey="true"
   concero:init="true" />
```

The interpretation of the mapping between the connector field (as specified by the Concero:tafield attribute) and the resource field (as specified by the Concero:resfield attribute) is determined by the connector. The LDAP connector has code to interpret the mappings in one way, as follows:

— The connector attribute names are specified in square braces, like this: [xyz]. The value of attribute xyz is taken from the UserModel during provisioning.

— Composite attributes can be specified in the LDAP connector mapping file. To do this, specify [attr1] xxxx [attr2] as the connector attribute. This specifies that the value of the attr1 and attr2 attributes should be combined with the string xxxx to form a mapping for the specified resource field. LDAP connector has code to handle these composite mappings.

- **<attributeDefinition>**

  Defines the properties of each object's attribute. For example, the attribute definition for the HomeDir attribute defines that it must be between zero and 100 characters in length and can contain the following letters, numbers, and characters: a-z, A-Z, 0-9, @, +, and a space.

  Here is an excerpt from the `ActiveDir.xml` file:

```
<attributeDefinition name="HomeDir" description="User Home
directory" type="xsd:string" >
  <properties>
    <attr name="minLength">
      <value>0</value>
    </attr>
    <attr name="maxLength">
      <value>128</value>
    </attr>
    <attr name="pattern">
      <value><![CDATA[[a-zA-Z0-9@]+]]> </value>
    </attr>
  </properties>
</attributeDefinition>
```

- **<concero:entitlementMappingDefinition>**

  Defines how entitlements are mapped to users.

- **<concero:objectStatus>**

  Defines how to assign status to a user.

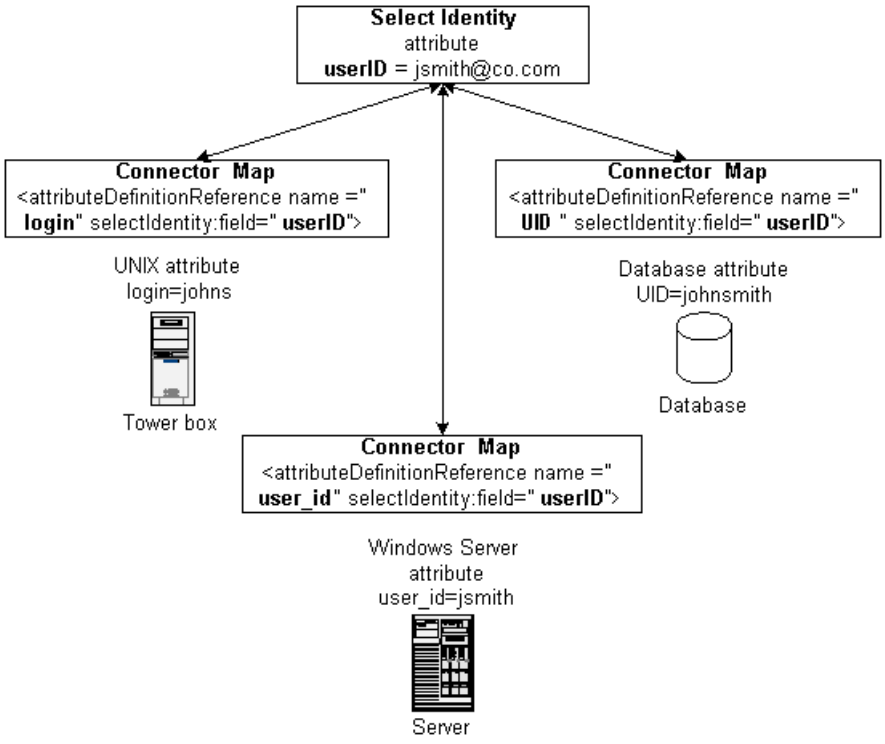
- **<concero:relationshipDefinition>**

  Defines how to create relationships between users.

Creating a Mapping File

Create a mapping file that maps each attribute on the physical resource to an attribute on the connector. (To complete this mapping, attributes must be created using the Select Identity client to map a name on the server to this name on the connector.) For example, the connector may store the user ID in a field called **userID** and the resource may store the ID in a field called **user_id**. The connector will reference the mapping file to understand the target field on the resource for each user value.

The following illustrates the relationship between the fields in Select Identity, the connector, and the resource:



Instances of UserModel and GroupModel are populated and provided by Select Identity when it calls the TAConnector methods. Obtain user and group attributes from here and map them to the resource using map file.

You determine the format of the mapping file. The connector may require only a simple mapping stored in a text file. Here is a simple text file example where the Select Identity field is specified first and a pipe (|) separates the fields:

```
User Name|UserId
Password|Password
User Name|cn
First Name|givenName
Last Name|sn
[First Name] [Last Name]|displayName
Title|Title
```

```
Directory|homeDirectory
Email|Mail
Address 1|streetAddress
```

Or, the connector may require a format that supports robust mapping, such as an XML file. XML mapping files are used by all connectors built and provided by HP. Here is an excerpt from the `iPlanet.xml` file, which is provided with the LDAP connector. Refer to Mapping File Overview on page 31 for a full description of the file.

```
<objectClassDefinition name="User" description="LDAP User">
  <properties>
    <attr name="CREATE">
      <value>true</value>
    </attr>
    <attr name="READ">
      <value>true</value>
    </attr>
    <attr name="UPDATE">
      <value>true</value>
    </attr>
    <attr name="DELETE">
      <value>true</value>
    </attr>
    <attr name="ENABLE">
      <value>true</value>
    </attr>
    <attr name="DISABLE">
      <value>true</value>
    </attr>
    <attr name="RESET_PASSWORD">
      <value>true</value>
    </attr>
    <attr name="EXPIRE_PASSWORD">
      <value>false</value>
    </attr>
    <attr name="CHANGE_PASSWORD">
      <value>true</value>
    </attr>
  </properties>
  <memberAttributes>
    <!-- For iPlanet -->
    <attributeDefinitionReference name="UserName" required="true"
     concero:tafield="[UserName]" concero:resfield="uid"
     concero:isKey="true" concero:init="true"/>
    <attributeDefinitionReference name="Password" required="false"
     concero:tafield="[Password]" concero:resfield="userpassword"
     concero:init="true" />
```

Installing a Connector

To install the connector on the Select Identity server, you must copy the connector files to the target locations and configure the application server. The following steps provide general guidelines for installing a connector; the details will depend on how the connector was implemented and the type of application server.

1. If necessary, stop the application server.
2. Copy the *connector*.rar file and the mapping file into the Select Identity folder on the application server (which should have been created when Select Identity was installed).
3. Start the application server.
4. Edit the startweblogic.cmd file to specify the location of the mapping file. The startweblogic.cmd file resides in the *WebLogic_home*/user_projects/domains/*domain*/ directory on WebLogic 8.1.

   Locate the line set CLASSPATH=%CLASSPATH% in the startweblogic.cmd file and add C:\Select_Identity*file_name* to the class path.
5. Restart the application server.

   After you install the connector the first time, you do not need to restart the application server if the *connector*.rar and mapping files change. This is because neither the *connector*.rar file nor name of the mapping file were added to the classpath.
6. Deploy the connector on the application server, as follows:
   a. On the WebLogic Administrator Console, navigate to ***My_domain* –> Deployments –> Connector Modules**.
   b. Click **Deploy a New Connector Module**.
   c. Locate and select the *connector*.rar file (in the Select_Identity directory).
   d. Click **Target Module**.
   e. Select the **My Server** (which is your server instance) check box.
   f. Click **Continue**. Review your setings.

g Keep all default settings and click **Deploy**.

h Restart the application server.

7 *Two-way connector only*
Install and configure the agent on the resource with which the connector communicates to provision users. The agent may also be used to synchronize changes to the identity objects, pushing the changes from the resource to Select Identity. The installation and configuration steps are agent-specific, though you can refer to the *HP OpenView Select Identity Installation Guide for the NTLocal and NTDomain Connector* for an example, if you purchased this connector.

Deploying a Connector in Select Identity

After you create a connector, you can deploy it using the Select Identity interface. The following provides an overview of the procedures you must complete in order to deploy your connector:

1 After you build and install the connector, you must register it with Select Identity. Do so on the home page of the Connector Management tab by clicking the **Deploy New Connector** button. Complete this procedure, referencing your connector files, as described in the "Connector Management" chapter of the *HP OpenView Select Identity Administrator Guide*.

2 You must deploy the resource that uses the newly created connector. On the home page of the Resource Management tab, click the **Deploy New Resource** button. Complete the steps in this procedure, referencing the new connector created in step 1, as described in the "Resource Management" chapter of the *HP OpenView Select Identity Administrator Guide*.

3 Create attributes that link Select Identity to the connector. For each mapping in the connector's mapping file, create an attribute using the Attributes capability on the Select Identity client. Refer to the "Attributes" chapter in the *HP OpenView Select Identity Administrator Guide* for more information.

4 Create a Service that will use the newly created resource. To do so, click the **Deploy New Service** button on the home page of the Service Management tab. Complete this procedure as described in "Service

Management" of the *HP OpenView Select Identity Administrator Guide*. You will reference your new resource created in step 2 while creating this service.

Testing a Connector

To test a connector, verify that you can perform user provisioning tasks. Perform each of the following tasks to thoroughly test the connector.

1. Verify provisioning operations using the Select Identity client. Go to the Users home page and perform the following tasks, if applicable. Refer to the *HP OpenView Select Identity Administrator Guide* for detailed information.
   - Add a user
   - Modify the user attributes
   - Delete an existing user from the resource
   - Retrieve the details of user from the resource
   - Disable the user on the resource
   - Enable the user on the resource
   - Change the user's password
   - Retrieve all entitlements present in the resource
   - Associate entitlements with an existing user on the resource
   - Remove entitlements from the user
   - Synchronize passwords, which involves changing a user's password on the resource; the resource should then propagate to the existing user in Select Identity
   - With an agent-based connector, an SPML **<extendedRequest>** request should be sent to the Select Identity Web Service with the password information
   - Reverse synchronization, which involves synchronizing Select Identity with changes to identity information on the resource.

2. Perform the following operations on the resource directly using its interface. These tests involve verifying the reconciliation in Select Identity. With an agent-based connector, SPML requests should be sent back to the Select Identity Web Service with the changes made on the resource.
   - Add a new user on the resource. This should result in an SPML **<addRequest>** request including all the attributes of the user.
   - Modify the user attributes on the resource. This should result in an SPML modifyRequest with the modified attribute information
   - Delete an existing user from the resource. This should result in an SPML deleteRequest with the id of the user
   - Disable the user on the resource. This should result in an SPML extendedRequest with all the attributes of the user
   - Enable the user on the resource. This should result in an SPML extendedRequest with all the attributes of the user
   - Associate entitlements to an existing user on the resource. This should result in an SPML modifyRequest with the new entitlements added.
   - Dissociate entitlements from the user. This should result in an SPML modifyRequest with the removal of entitlements
   - Associate some and dissociate some entitlements on the user on the resource. This should result in an SPML modifyRequest addition/ deletion of entitlements.
3. Verify changes made on the ID object in the Select Identity repository. You can view user attribute or service membership information in the repository.

3

LDAP Connector Example

The LDAP connector enables HP OpenView Select Identity to manage user data in LDAP. It is a one-way connector and pushes changes made to user data in the Select Identity database to a target LDAP server. This connector is generic and can be used to connect to any LDAP data source. The mapping file controls how Select Identity fields are mapped to LDAP fields.

The mapping file, source files, definition file, and build files reside in a directory with the following structure in the `docs/connectors` directory on the Select Identity CD:

```
ldapv3/
  com/
    trulogica/
      truaccess/
          connector/
            ldap/
              ldapv3/
                LDAPConnector.java
                LDAPConnectorFactory.java
                LDAPManagedConnection.java
                LDAPManagedConnectionFactory.java
                LDAPManagedConnectionMetaData.java
                LDAPParamFactory.java
                LDAPParamResources.properties
                LDAPParamValueBean.java
                LDAPRAMetaData.java
```

```
                LDAPUtil.java
            schema/
              spml/
                ActiveDir.xml
                CAEtrust.xml
                iPlanet.xml
  META-INF/
    ra.xml
    weblogic-ra.xml
  build.bat
  build.properties
  build.xml
  build_common.xml
  build_rar.xml
```

This chapter provides an explanation of the source code that implements the LDAP connector, the mapping file that Select Identity refers to when pushing data, and the packaging. Use this example to help you build your own connector.

Description of Source Files

The source files for the LDAP connector are provided on the Select Identity CD, in the `docs/connectors/LDAPv3` directory. The following provides a description of the files:

- `LDAPConnector.java`

  This is the implementation of TAConnector interface to provision users onto the LDAP data store. This represents a physical connection to the LDAP store.

  The class uses the JNDI API for a directory interface to access and update LDAP. Connection parameters should contain the URL to access the LDAP store and the root directory name and password. This class uses the SPML-based XML mapping file to map Select Identity resource fields to LDAP attributes.

- `LDAPManagedConnectionFactory.java`

  This is the implementation of the javax.resource.spi.ManagedConnectionFactory interface. This class is registered with the application server by specifying the managedconnectionfactory-class in the `ra.xml` deployment descriptor file.

  The application server calls on this implementation to create and return an instance of ManagedConnection, which represents the connection to the resource and matches existing managed connections with the given one. Also, the connector parameter factory implementation is registered with JNDI in this file.

- `LDAPConnectorFactory.java`

  This is the implementation of TAConnectorFactory interface and represents a factory to create managed connections. This class is registered with the application server by specifying the factory under the connectionfactory-impl-class in the `ra.xml` file.

  The getConnection(TAConnectorParamValueBean connParam) method is implemented and it calls on the application server connection manager to allocate and return a new connection.

- `LDAPManagedConnection.java`

  This is the implementation of the javax.resource.spi.ManagedConnection interface and it represents the physical connection to the resource.

  The application server calls the getConnection() method in this class to get a connection handle to the resource. The connection parameter value bean is passed in by the application server. A local copy of this bean is created and a new instance of LDAPConnector is created and returned.

  A copy of the schema repository is maintained here for reference by LDAPConnector. This repository is built from the mapping file.

- `LDAPParamFactory.java`

  This is the implementation of TAConnectorParameterFactory interface. It is instantiated and registered with the JNDI so that Select Identity can lookup and call on this instance to create instances of beans that contain the connection parameter values.

- `LDAPParamValueBean.java`

  This is the derived class of the TAConnectorParamValueBean abstract class. It contains the names of all of the connection parameters needed to connect to and access the LDAP resource, as follows:

  - accessURL — the URL to access the LDAP store
  - suffix — the suffix of the domain name (DN) for all users and groups
  - rootDN — the root DN to log in to the LDAP store
  - rootPassword — the root password
  - userSuffix — the user suffix, such as ou=Users
  - userObjectClass — the Object class of all users
  - groupSuffix — the group suffix, such as ou=Groups
  - groupObjectClass — the Object class of all group objects
  - mappingFile — the name of the file that contains the attribute mappings

  Each instance of this bean contains one set of information for the connection parameters.

  Also, the following method are implemented:

  - getParamNames() returns all the above-listed connection parameters
  - get(name) returns the value of the connection parameter
  - set(name, value) stores the value of the connection parameter. This value is passed from the configuration at the time of resource deployment

- `LDAPManagedConnectionMetaData.java`

  This is the implementation of the javax.resource.spi.ManagedConnectionMetaData interface and is used to return the EIS product name, version, and maximum connections allowed to the application server.

- `LDAPRAMetaData.java`

  This is the implementation of the javax.resource.cci.ResourceAdapterMetaData interface and is used to return the resource adapter-specific information to the application server, such as the adapter name, vendor name, and version.

- `LDAPUtil.java`

  This is a utility class that implements some methods used by other parts of the connector.

- `LDAPParamResources.properties`

  This is a text file containing configuration properties for the connector and has the default values for all of the connection parameters. This file is read in during startup by `LDAPUtil.java` to return the default values of the connection parameters. These are displayed in the Select Identity client, on the Resources home page.

- `ra.xml`

  This is the deployment descriptor for the resource adapter implementing the connector. The interface and implementation class names are registered here.

  As described in Step 6 on page 20, this file contains the name of the connector, the configuration, the interface names of the connector, and the JNDI name for the connector. Refer to the `ra.xml` file provided by the LDAP connector as an example when creating your own. Create this XML file according to the JCA specification. Here is an explanation of the elements in the `ra.xml` file:

  - **<display-name>**, **<vendor-name>**, **<spec-version>**, **<eis-type>**, **<version>**, and **<license>**

    Provides general information about the connector.

  - **<managedconnectionfactory-class>**

    Specifies the path to the class implementing the ManagedConnectionFactory interface.

  - **<connectionfactory-interface>**

    Specifies the path to the TAConnectorFactory interface.

  - **<connectionfactory-impl-class>**

    Specifies the path to the class implementing the TAConnectorFactory interface.

  - **<connection-interface>**

    Specifies the path to the TAConnector interface.

- **<connection-impl-class>**

  Specifies the path to the class implementing the TAConnector interface.

- **<transaction-support>**

  Specifies whether the connector supports transactions.

- **<config-property>**

  Defines a configuration property for the connector. For example, the UserName property is defined. It is a string and its value is set to cn=Directory Manager.

  A <config-property> element is defined for each of the connector's configuration properties.

- **<reauthentication-support>**

  Specifies whether the connector supports authentication after the connector has communicated with Select Identity.

- `weblogic-ra.xml`

  This is the WebLogic-specific deployment descriptor for the resource adapter and contains the LDAP connector JNDI name.

Description of Build Files

The following XML and property files are used by Apache Ant to build the LDAP connector. Refer to the LDAP files on the CD to view the contents.

- `build.xml`

  This is the main build file for the connector. It references the `build.properties` file and calls the `build_rar.xml` file, which contains information about building the `.rar` file. It also calls the `build_common.xml` file to build common files.

- `build_common.xml`

  This file builds the common classes.

- `build_rar.xml`

  This file contains information about building the `.rar` file.

- `build.properties`

  This file contains definitions used by the build files.

- `build.bat`

  This file launches the build using Ant.

glossary

A

Access Control List (ACL)

An abstraction that organizes entitlements and controls authorization. An ACL is list of entitlements and users that is associated with a secured object, such as a file, an operation, or an application. In an ACL-based security system, protected objects carry their protection settings in the form of an ACL.

Access Management

The process of authentication and authorization.

Action

An action represents a task that can be performed within each Select Identity capability.

See also: ***capability***

Admin Role

A template that defines the administrative actions that can be performed by a user. An Administrative Service is created to provide access to roles. Users are then given access to the Service. Users with administrative roles can also grant their set of roles to another administrator within their Service context.

Approval Process

The process of approving the association, modification, or revocation of entitlements for an identity. This process is automated of these through workflow templates.

Approver

A Select Identity administrator who has been given approval actions through an Admin Role.

Attribute

An attribute is an individual field that helps define an identity profile. For each identity, an attribute has a corresponding value. For example, an attribute could be “department” with possible values of “IT,” “sales,” or “support.”

Audit Report

A report that provides regular account interaction information within the Select Identity system.

Authentication

Verification of an identity’s credentials.

Authoritative Source

A resource that has been designated as the “authority” for identity information. Select Identity accounts can be reconciled against accounts in an authoritative source.

Authorization

Real-time enforcement of an identity’s entitlements. Authentication is a prerequisite for authorization.

Auto Discovery

The process of adding user accounts to the Select Identity system for a specified Service through the use of a data file.

B

Business Relationship

A Select Identity abstraction that defines how a logical grouping of users will access a Select Identity Service. The Select Identity Service is a superset of all the identity management elements of a business service.

Business Service

A business service is a product or facility offered by, or a core process used by, a business in support of its day-to-day operations. Example business services could include an online banking service, the customer support process, and IT infrastructure services such as email, calendaring, and network access.

See also: ***service***

C

Capability

Actions that can be performed within the Select Identity client are grouped by capability, or link, in the interface.

See also: ***action***

Challenge and Response

A method of supplying alternate authentication credentials, typically used when a password is forgotten. Select Identity challenges the end user with a question and the user must provide a correct response. If the user answers the question correctly, Select Identity resets the password to a random value and sends email to the user. The challenge question can be configured by the administrator. The valid response is stored for each user with the user's profile and can be updated by an authenticated user through the Self Service pages.

Configurations

The Configurations capability enables you to import and export Select Identity settings and configurations. This is useful when moving from a test to a production environment.

Configuration Reports

Configuration reports provide current system information for user, administrator, and Service management activities.

Connector

A J2EE connector that communicates with the system resources that contain your identity profile information.

Context

A Select Identity concept that defines a logical grouping of users that can access a Service.

Contextual Identity Management (CIM)

An organizational model that introduces new abstractions that simplify and provide scale to the business processes associated with identity management. These abstractions are modeled after elements that exist in businesses today and include Select Identity Services and Business Relationships.

Credentials

A mechanism or device used to verify the authenticity of an identity. For example, a user ID and password, biometrics, and digital certificates are considered credentials.

D

Data File

An SPML file that enables you to define user accounts to be added to Select Identity through Auto Discovery or Reconciliation.

Delegated Administration

The ability to securely assign a subset of administrative roles to one or more users for administrative management and distribution of workload. Select Identity enables role delegation through the Self Service pages from one administrator to another user within the same Service context.

Delegated Registration

Registration performed by an administrator on behalf of an end user.

E

End User

A role associated to every user in the Select Identity system that enables access to the Self Service pages.

Entitlement

An abstraction of the resource privileges granted to an identity. Entitlements are resource-specific and can be resource account IDs, resource role memberships, resource group memberships, and resource access rights and privileges. Entitlements are also considered privileges, permissions, or access rights.

External Call

A programmatic call to a third-party application or system for the purpose of validating accounts or constraining attribute values.

F

Form

An electronic document used to capture information from end users. Forms are used by Select Identity in many business processes for information capture and system operation.

I

Identity

The set of authentication credentials, profile information, and entitlements for a single user or system entity. Identity is often used as a synonym for “user,” although an identity can represent a system and not necessarily a person.

Identity Management

The set of processes and technologies involved in creating, modifying, deleting, organizing, and auditing identities.

M

Management

The ongoing maintenance of an object or set of objects, including creating, modifying, deleting, organizing, auditing, and reporting.

N

Notifications

The capability that enables you to create and manage templates that define the messages that are sent when a system event occurs.

P

Password Reset

The ability to set a password to a system-generated value. Select Identity uses a challenge and response method to authenticate the user and then allow the user to reset or change a password.

Policy

A set of regulations set by an organization to assist in managing some aspect of its business. For example, policy may determine the type of internal and external information resources that employees can access.

Process

A repeatable procedure used to perform a set of tasks or achieve some objective. Whether manual or automated, all processes require input and generate output. A process can be as simple as a single task or as complicated a multi-step, conditional procedure.

See also: ***approval process***

Profile

Descriptive attributes associated with an identity, such as name, address, title, company, or cost center.

Provisioning

The process of assigning authentication credentials to identities.

R

Reconciliation

The process by which Select Identity accounts are synchronized with a system resource. Accounts can be added to the Select Identity system through the use of an SPML data file.

Registration

The process of requesting access to one or more resources. Registration is generally performed by an end user seeking resource access, or by an administrator registering a user on a user's behalf.

See also: ***delegated registration***, ***self registration***

Request

An event within the Select Identity system for the addition, modification, or removal of a user account. Requests are monitored through the Request Status capability.

Resource

Any single application or information repository. Resources typically include applications, directories, and databases that store identity information.

Role

A simple abstraction that associates entitlements with identities. A role is an aggregation of entitlements and users, typically organized by job function.

See also: ***administrative role***

Rule

A programmatic control over system behavior. Rules in Select Identity are typically used for programmatic assignment of Services. Rules can also be used to detect changes in system resources.

S

Self Registration

Registration performed by an end user seeking access to one or more resources.

Self Service

The ability to securely allow end-users to manage aspects of a system on their own behalf. Select Identity provides the following self-service capabilities: registration, profile management, and password management (including password change, reset, and synchronization).

Service

A business-centric abstraction representing resources, entitlements, and other identity-related entities. Services represent the products and services that you offer to customers and partners.

Service Attribute

A set of attributes and values that are available for or required by a Service. Attributes are created and managed through the Attributes pages.

See also: ***Attributes***

Service View

A restricted view of a Service that is valid for a group of users. Views enable you to define a subset of Service registration fields, change field names, reorder fields, and mask field values for specific users.

Single Sign-On (SSO)

A session/authentication process that permits a user to enter one set of credentials (name and password) in order to access multiple applications. A Web SSO is a specialized SSO system for web applications.

SPML Data File

A file that is used to add and provision accounts within Select Identity.

See also: ***Data File***

U

Users

The Select Identity capability that provides consistent account creation and management across Services.

W

Workflow

The tasks, procedural steps, organizations or people involved, and required input and output information needed for each step in a business process. In identity management, the most common workflows are for provisioning and approval processes.

Workflow Engine

A system component that executes workflows and advances them through their flow steps.

Workflow Studio

The Select Identity capability that enables you to create and manage workflow templates.

index

R

S

T

X