Oracle® Fusion Middleware

Developer's Guide for Identity Governance Framework

11*g* Release 1 (11.1.1)

E16588-03

March 2013

This guide provides an introduction to the Identity Governance Framework (IGF) initiative that enables secure exchange of identity-related information between users and applications and service providers. It describes how to develop with the Identity Governance Framework based developer APIs Oracle has made available.

ORACLE®

Oracle Fusion Middleware Developer's Guide for Identity Governance Framework, 11*g* Release 1 (11.1.1)

E16588-03

Contents

3   Using the ArisID API

4   Migrating to Identity Directory API

List of Examples

List of Figures

List of Tables

Preface

This guide provides an introduction to Identity Governance Framework and describes how to use the related developer APIs Oracle has made available. It describes the Identity Directory API, which is a common service for identity management applications to access and manage identity information.

Audience

This document is intended for developers who are writing applications that use the Oracle Fusion Middleware Identity Governance Framework based APIs.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Fusion Middleware Identity Governance Framework Identity Directory Java API Reference*

- *Oracle Fusion Middleware Identity Governance Framework IDXUserRole Java API Reference*

- *Oracle Fusion Middleware Identity Governance Framework UserRole Java API Reference*

- *Oracle Fusion Middleware Application Security Guide*

- Javadocs for Project Aristotle - ArisID Attribute Services, at: http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New

The new features of the Identity Governance Framework based Oracle APIs are listed here, for both the present release and the previous releases.

New Features for 11gR1 PS 6 (11.1.1.7)

11*g* Release 1 (11.1.1) provides the Identity Directory API, a common service for identity management applications to access and manage identity information. The service can be used in both Java EE and Java SE modes. For more information, see Chapter 2, "Using the Identity Directory API". For information about migration to the Identity Directory API, see Chapter 4, "Migrating to Identity Directory API."

1

Introduction to Identity Governance Framework

The Identity Governance Framework (IGF) initiative enables secure exchange of identity-related information between users and applications and service providers. It provides privacy and governance semantics to applications and services infrastructure. This chapter provides an introduction to the Identity Governance Framework and the related developer APIs Oracle has made available. This chapter contains the following topics:

- Section 1.1, "About the Identity Governance Framework"
- Section 1.2, "About the Identity Governance Framework APIs"
- Section 1.3, "System Requirements and Certification"

1.1 About the Identity Governance Framework

The Identity Governance Framework is an open initiative designed to meet the following goals:

- To simplify the development of identity information access regardless of where that information is stored.
- To simplify the management (also known as **governance**) of how applications use identity data, in particular, sensitive data.

As part of this initiative, Oracle has contributed key initial specifications and is making them available to the community. These specifications provide a common framework for defining usage policies, attribute requirements, and developer APIs pertaining to the use of identity related information. These enable businesses to ensure full documentation, control, and auditing regarding the use, storage, and propagation of identity-related data across systems and applications.

This section contains the following topics:

- Benefits to Organizations
- Benefits to Developers

1.1.1 Benefits to Organizations

Organizations need to maintain control and integrity of sensitive personal information about their customers, employees, and partners. Data related to social security numbers, credit card numbers, medical history and more are increasingly under scrutiny by regulations seeking to prevent abuse or theft of such information. Privacy conscious organizations frequently have reacted to these requirements by enforcing

overly strict controls and processes that hinder business operations and impact productivity, flexibility, and efficiency. At the opposite end of the spectrum, some organizations do not take the care needed to safeguard this information, potentially putting identity-related data at risk without sufficient oversight and control. The Identity Governance Framework enables a standards-based mechanism for enterprises to establish "contracts" between their applications so that identity related information can be shared securely and with confidence that this data will not be abused, compromised, or misplaced. Using this framework, organizations have complete visibility into how identity information is stored, used, and propagated throughout their business. This enables organizations to automate controls to streamline business processes without fear of compromising the confidentiality of sensitive identity related information.

1.1.2 Benefits to Developers

The Identity Governance Framework is an agreed-upon process for specifying how identity-related data is treated when writing applications. This provides developers a standards-based way to easily write applications that use this data so that governing policies can be used to control it. This will result in faster development of privacy aware applications.

IGF enables the decoupling of identity-aware applications from a specific deployment infrastructure. Specifically, using IGF enables developers to defer deciding how identity related information will be stored and accessed by their application. Developers do not need to worry about whether they should use a SQL database, an LDAP directory, or other system. In the past, developers were forced to write highly specific code, driving technology and vendor lock-in.

For example, The Identity Directory API provides methods for accessing and managing identity information in a directory server that is the domain identity store. Entity definitions, entity relationships, and the physical identity store details can be configured using either the Identity Directory Configuration APIs or Mbeans. The Identity Directory API is used to initialize the Identity Directory Service. The Identity Directory Service provides an interface to both access and modify users and group information from different identity stores. For more information, see Chapter 2, "Using the Identity Directory API".

Another example is the ArisID API, which handles the hard work of data retrieval, transformation, and policy-enforcement when it comes to identity-based information. By using a Client Attribute Requirement Markup Language (CARML) file and declarations, applications will support flexible deployment in a wide range of environments without the need for ongoing specialized developer enhancements. For more information, see Chapter 3, "Using the ArisID API".

1.2 About the Identity Governance Framework APIs

Oracle has made the following APIs available that are based on the Identity Governance Framework:

- **Identity Directory API**

    The Identity Directory API is a common service for identity management applications to access and manage identity information. The service can be used in both Java EE and Java SE modes. For more information, see Chapter 2, "Using the Identity Directory API".

- ArisID **API**

The ArisID API provides enterprise developers and system architects a library for building identity-enabled applications using multiple identity protocols. The ArisID API enables developers to specify requirements for identity attributes, roles, and search filters by using Client Attribute Requirements Markup Language (CARML). For more information, see Chapter 3, "Using the ArisID API".

1.3 System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-requirements-100147.html

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html

2

Using the Identity Directory API

This chapter describes the architecture and key functionality of the Identity Governance Framework Identity Directory API (Identity Directory API) and Identity Directory Service. The Identity Directory API supports accessing and managing users, groups, organizations, and can be extended to support new entity types with relationships defined between these entities. This chapter contains the following topics:

- Section 2.1, "About Identity Directory API"
- Section 2.2, "Configuring Identity Directory API"
- Section 2.3, "Design Recommendations"
- Section 2.4, "Examples"

2.1 About Identity Directory API

The Identity Directory API provides a service for identity management applications to access and manage identity information. The API is flexible and fully configurable by clients supporting heterogeneous identity stores having standard and specific schemas, and is robust with both high-availability and failover support.

The API uses the Identity Governance Framework and provides all the benefits of Identity Governance. The API can be used in both Java EE and Java SE modes. For more information about the Identity Governance Framework, see Chapter 1, "Introduction to Identity Governance Framework".

The API supports the following actions:

- Create/Read/Update/Delete (CRUD) operations on User, Group, Org, and generic entities
- Get operation on User Account State
- Identity Directory API configuration sharing
- Support for directory servers such as Oracle Internet Directory, Oracle Universal Directory, Oracle Directory Server EE, and AD.

Identity Directory Service consists of the following:

- **Identity Directory API**

  The Identity Directory API provide methods for accessing and managing identity information in a directory server that is the domain identity store. Entity definitions, entity relationships, and the physical identity store details can be configured using either the Identity Directory Configuration APIs or Mbeans. Directory service instance capabilities can be queried using getter methods.

■ **Identity Directory API Configuration**

Identity Directory API configuration comprises logical entity configuration and physical identity store configuration.

2.1.1 About the Identity Directory Service

The Identity Directory Service is a common service used by identity management products to access and manage an Identity Directory. The Identity Directory API is used to initialize the Identity Directory Service. The Identity Directory Service provides an interface to both access and modify users and group information from different identity stores. An Identity Directory is an instance of the Identity Directory Service having:

■ a unique name (IDS name)

■ a logical entity configuration

■ a physical identity store configuration

2.1.2 Identity Directory Service Architecture

Figure 2–2 shows the logical architecture of the Identity Directory API.

Figure 2–1   Identity Directory API Architecture
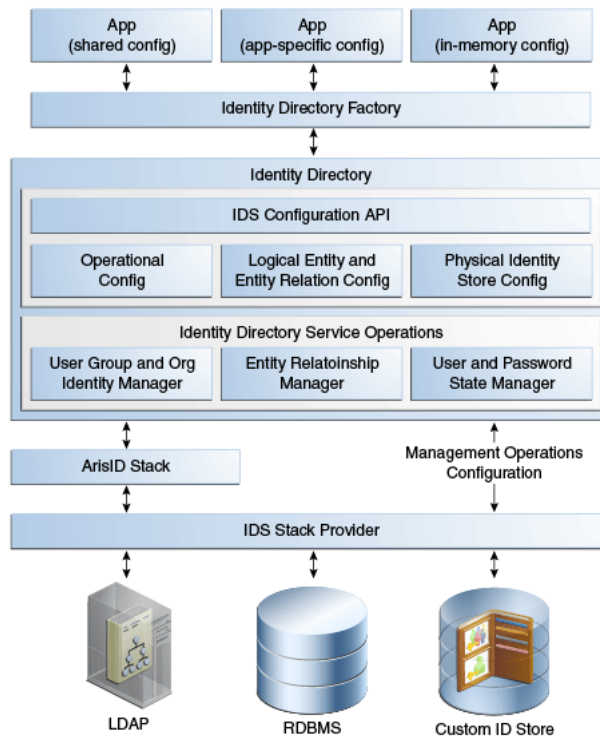
Figure 2–2 shows the relationship between the Identity Directory API components.

Figure 2–2   Identity Directory API Components
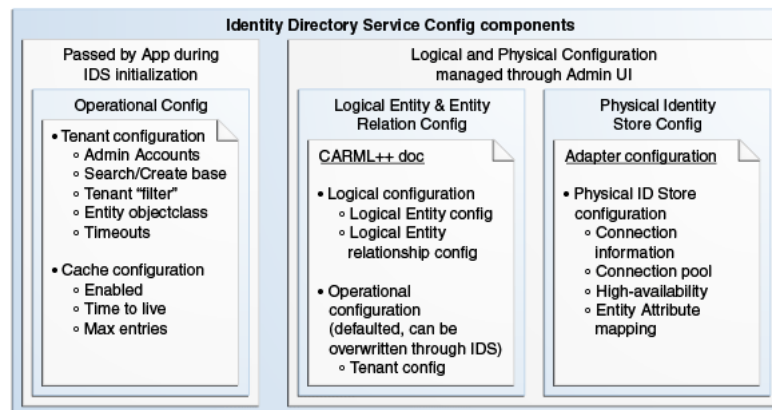
2.2 Configuring Identity Directory API

The Identify Directory API provides an interface to access and modify users and group information from different identity stores.

The Identity Directory Service configuration is a combination of the logical entity configuration, the physical identity store configuration, and operational configuration.

The logical entity configuration and operational configuration is stored in ids-config.xml. This file is located in the same directory as jps-config.xml. For example, in a Java EE environment the location is:

DOMAIN_HOME/config/fmwconfig/ids-config.xml

The physical identity store configuration is stored in ovd/ids/adapters.os.xml. For example, in a Java EE environment the ovd directory is located in:

DOMAIN_HOME/config/fmwconfig

This section contains the following topics:

- Logical Entity Configuration

- Physical Identity Store Configuration

- Operational Configuration

2.2.1 Logical Entity Configuration

Table 2–1 describes the logical entity configuration properties.

Table 2–1   Logical Entity Configuration Properties

Name	Description
name	Name that uniquely identifies the Identity Directory Service.
description	Detailed description of the Identity Directory Service.
ovd.context	Valid values are default or ids. Use default for connecting to the same identity store configured in OPSS. Use ids to connect to any physical identity store configured independent of OPSS. Only out-of-the-box identity directories, that is userrole and idxuserrole, use default value.
app.name	Optional property to specify the specific application for which the Identity Directory Service is being configured.

Table 2–2 describes the logical attributes.

Table 2–2   Logical Entity Attributes

Name	Description
name	Logical attribute name.
dataType	Valid data type values are as follows: string, boolean, integer, double, datetime, binary, x500name, and rfc822name.
description	Detailed description of the logical attribute.
readOnly	Default is false. Use true if the attribute is read-only.
pwdAttr	Default is false. Use true if the attribute is a password attribute.

Table 2–3 describes the properties required in each logical entity definition.

Table 2–3   Logical Entity Definition Properties

Name	Description
name	Name of the entity.
type	Valid entity values are as follows: user, group, org and other.
idAttr	Logical attribute that uniquely identifies the entity.
create	Use true if creating this entity is allowed. Use false otherwise.
modify	Use true if modifying this entity is allowed. Use false otherwise.
delete	Use true if deleting this entity is allowed. Use false otherwise.
search	Use true if search of this entity to be allowed. Use false otherwise.
Attribute References	List of entity attribute references that contain the following details: ■ name: Logical attribute name. ■ defaultFetch: Default value is true. Use true if the attribute will be fetched by default. For example, when the entity is read using Identity Directory API, this attribute value is fetched from the identity store even though this attribute is not included in the requested attributes. ■ filter: Search filter type with one of the following valid values: none, dynamic, equals, notequals, beginswith, contains, doesnotcontain, endswith, greaterequal, lessequal, greaterthan, and lessthan. Value none means no filter support.

Table 2–4 describes the properties required in each logical entity relationship definition.

Table 2–4   Logical Entity Relationship Properties

Name	Description
name	Name of the entity relationship.
type	Valid entity values are as follows: OneToOne, OneToMany, ManyToOne, and ManyToMany.
fromEntity	Name of the first entity in the Entity Relationship.

Table 2–4   (Cont.) Logical Entity Relationship Properties

Name	Description
fromAttr	The first entity's attribute. Value of this attribute relates to the second entity in the relationship.
toEntity	Name of the second entity in the Entity Relationship.
toAttr	The second entity's attribute. Value of the fromAttr property maps to this attribute in second entity.
recursive	Use true if the entity relationship is recursive. Default is false.

2.2.2 Physical Identity Store Configuration

Table 2–5 describes the physical identity store configuration properties.

Table 2–5   Physical Identity Store Configuration Properties

Name	Description
Host and Port	Host and Port information of the Identity Store. Alternate Host and Port details also can be setup for failover.
Directory Type	Type of directory. Valid values are: OID, ACTIVE_DIRECTORY, IPLANET, EDIRECTORY, OPEN_LDAP, WLS_OVD, and OUD.
Bind DN and Password	Credentials to connect to the directory.

2.2.3 Operational Configuration

The operational configuration contains mainly base, name attribute, and objectclass configuration for each of the entities.

Table 2–6 describes the operational configuration entities.

Table 2–6   Operational Configuration Entities

Name	Description
entity.searchbase	Container under which the entity should be searched.
entity.createbase	Container where the new entity will be created.
entity.name.attr	RDN attribute of the entity.
entity.filter.objclasses	The objectclass filters to be used while searching this entity.
entity.create.objclasses	The objectclasses to be added while creating this new entity.

2.3 Design Recommendations

This section contains the following topics:

- Keep the defaultFetch Attributes Minimal

- Initialize the Identity Directory Once

2.3.1 Keep the defaultFetch Attributes Minimal

While configuring a new Identity Directory, try to keep the number of entity defaultFetch attributes minimal. Also, try to have large attributes like jpegphoto configured with a defaultFetch value of false. The reason is every time the entity is read from the backend, all the defaultFetch attributes from backend directory will be retrieved. Too many defaultFetch attributes will affect the performance.

2.3.2 Initialize the Identity Directory Once

Initialization of Identity Directory has some overhead to initialize the entire ArisId stack. As a result, applications should initialize the Identity Directory once, preferably on application startup, and use only one handle throughout.

2.4 Examples

This section provides the following code samples:

- Initialize and Obtain Identity Directory Handle
- Initialize and Obtain Identity Directory Handle from JPS Context
- Initialize and Obtain In-Memory Identity Directory Handle
- Add User
- Get User for Given Principal
- Modify User
- Get User for Given ID Value
- Search Users Using Complex Search Filter
- Change User Password
- Reset User Password
- Authenticate User
- Delete User
- Create Group
- Search Groups
- Get Management Chain
- Get Reportees of a User
- Add a Member to a Group
- Delete a Member From Group
- Get All The Groups User is a Member

2.4.1 Initialize and Obtain Identity Directory Handle

The following code sample initializes and obtains a handle to the identity directory:

```
import oracle.igf.ids.UserManager;
import oracle.igf.ids.GroupManager;
import oracle.igf.ids.config.OperationalConfig;
import oracle.igf.ids.IdentityDirectoryFactory;
import oracle.igf.ids.IdentityDirectory;
import oracle.igf.ids.IDSException;


public class IdsSample {

    private IdentityDirectory ids;
    private UserManager uMgr;
    private GroupManager gMgr;

    public IdsSample() throws IDSException {
```

```
            // Set Operational Config
            OperationalConfig opConfig = new OperationalConfig();

            // Set the application credentials (optional).  This
 overrides the credentials set in
            // physical ID store configuration
opConfig.setApplicationUser("cn=user1,dc=us,dc=example,dc=com");
opConfig.setApplicationPassword("password".toCharArray());

            // Set search/crate base, name, objclass, etc. config
 (optional).  This overrides default operational configuration
 in IDS
            opConfig.setEntityProperty("User", opConfig.SEARCH_BASE,
 "dc=us,dc=example,dc=com");
            opConfig.setEntityProperty("User", opConfig.CREATE_BASE,
 "dc=us,dc=example,dc=com");
            opConfig.setEntityProperty("User", opConfig.FILTER
_OBJCLASSES, "person");
            opConfig.setEntityProperty("User", opConfig.CREATE
_OBJCLASSES, "inetorgperson");
            opConfig.setEntityProperty("Group", opConfig.SEARCH
_BASE, "cn=groups,dc=us,dc=example,dc=com");
            opConfig.setEntityProperty("Group", opConfig.CREATE
_BASE, "cn=groups,dc=us,dc=example,dc=com");
            opConfig.setEntityProperty("Group", opConfig.FILTER
_OBJCLASSES, "groupofuniquenames");
            opConfig.setEntityProperty("Group", opConfig.CREATE
_OBJCLASSES, "groupofuniquenames");

            // Get IdentityDirectory "ids1" configured in IDS config
            IdentityDirectoryFactory factory = new
 IdentityDirectoryFactory();
            ids = factory.getIdentityDirectory("ids1", opConfig);

            // Get UserManager and GroupManager handles
            uMgr = ids.getUserManager();
            gMgr = ids.getGroupManager();
        }
    }
```

2.4.2 Initialize and Obtain Identity Directory Handle from JPS Context

The following code sample initializes and obtains the identity directory handle from
JPS context.

```
import oracle.igf.ids.UserManager;
import oracle.igf.ids.GroupManager;
import oracle.igf.ids.config.OperationalConfig;
import oracle.igf.ids.IdentityDirectoryFactory;
import oracle.igf.ids.IdentityDirectory;
import oracle.igf.ids.IDSException;

import oracle.security.jps.JpsContext;
import oracle.security.jps.JpsContextFactory;
import oracle.security.jps.service.idstore.IdentityStoreService;

public class IdsSample {

    private IdentityDirectory ids;
    private UserManager uMgr;
```

```
        private GroupManager gMgr;

    public IdsSample() throws IDSException {

        // Get IdentityDirectory from JpsContext
        try {
            JpsContext context =
JpsContextFactory.getContextFactory().getContext();
            IdentityStoreService idstore = (IdentityStoreService)
context.getServiceInstance(IdentityStoreService.class);
            ids = idstore.getIdentityStore();
        } catch (Exception e) {
            throw new IDSException(e);
        }

 // Get UserManager and GroupManager handles
        uMgr = ids.getUserManager();
        gMgr = ids.getGroupManager();
    }
}
```

2.4.3 Initialize and Obtain In-Memory Identity Directory Handle

The following code sample initializes and obtains the in-memory identity directory handle.

```
import java.util.ArrayList;
import java.util.List;

import oracle.igf.ids.UserManager;
import oracle.igf.ids.GroupManager;
import oracle.igf.ids.config.AttributeDef;
import oracle.igf.ids.config.AttributeRef;
import oracle.igf.ids.config.EntityDef;
import oracle.igf.ids.config.EntitiesConfig;
import oracle.igf.ids.config.EntityRelationship;
import oracle.igf.ids.config.IdentityStoreConfig;
import oracle.igf.ids.config.OperationalConfig;
import oracle.igf.ids.IdentityDirectoryFactory;
import oracle.igf.ids.IdentityDirectory;
import oracle.igf.ids.IDSException;

public class IdsSample {

    private IdentityDirectory ids;
    private UserManager uMgr;
    private GroupManager gMgr;
    public IdsSample() throws IDSException {

        // Add Attribute definitions
        List<AttributeDef> attrDefs = new ArrayList<AttributeDef>();
        attrDefs.add(new AttributeDef("cn", AttributeDef.DataType.STRING));
        attrDefs.add(new AttributeDef("firstname", AttributeDef.DataType.STRING));
        attrDefs.add(new AttributeDef("sn", AttributeDef.DataType.STRING));
        attrDefs.add(new AttributeDef("telephonenumber",
 AttributeDef.DataType.STRING));
        attrDefs.add(new AttributeDef("uid", AttributeDef.DataType.STRING));
        attrDefs.add(new AttributeDef("uniquemember",
 AttributeDef.DataType.STRING));
```

```
      // Add User entity definition
      List<EntityDef> entityDefs = new ArrayList<EntityDef>();
      EntityDef userEntityDef = new EntityDef("User", EntityDef.EntityType.USER,
"cn");
      userEntityDef.addAttribute(new AttributeRef("cn"));
      userEntityDef.addAttribute(new AttributeRef("firstname"));
      userEntityDef.addAttribute(new AttributeRef("sn"));
      userEntityDef.addAttribute(new AttributeRef("telephonenumber"));
      userEntityDef.addAttribute(new AttributeRef("uid"));
      entityDefs.add(userEntityDef);

      // Add Group entity definition
      EntityDef groupEntityDef = new EntityDef("Group",
EntityDef.EntityType.GROUP, "cn");
      groupEntityDef.addAttribute(new AttributeRef("cn"));
      groupEntityDef.addAttribute(new AttributeRef("uniquemember", false,
AttributeRef.FilterType.EQUALS));
      entityDefs.add(groupEntityDef);

      // Add Entity relationship definition
      List<EntityRelationship> entityRelations = new
ArrayList<EntityRelationship>();
      entityRelations.add(new EntityRelationship("user_memberOfGroup",
              EntityRelationship.RelationshipType.MANYTOMANY, "User",
"principal", "Group", "uniquemember"));
      entityRelations.add(new EntityRelationship("group_memberOfGroup",
              EntityRelationship.RelationshipType.MANYTOMANY, "Group",
"principal", "Group", "uniquemember", true));
      EntitiesConfig entityCfg = new EntitiesConfig(attrDefs,
entityDefs, entityRelations);


      // Create physical Identity Store configuration
      IdentityStoreConfig idStoreCfg = new IdentityStoreConfig(
          "ldap://host1:389,ldap://host2:389", "cn=orcladmin",
"password".toCharArray(), IdentityStoreConfig.IdentityStoreType.OID);

idStoreCfg.setHighAvailabilityOption(IdentityStoreConfig.HAOption.FAILOVER);
      idStoreCfg.setProperty(IdentityStoreConfig.HEARTBEAT_INTERVAL, "60");
      idStoreCfg.setProperty(IdentityStoreConfig.CONN_TIMEOUT, "30000");    //
milli sec
      idStoreCfg.setProperty(IdentityStoreConfig.MIN_POOLSIZE, "5");
      idStoreCfg.setProperty(IdentityStoreConfig.MAX_POOLSIZE, "10");
      idStoreCfg.setProperty(IdentityStoreConfig.MAX_POOLWAIT, "1000");     //
milli sec
      idStoreCfg.setProperty(IdentityStoreConfig.MAX_POOLCHECKS, "10");
      idStoreCfg.setProperty(IdentityStoreConfig.FOLLOW_REFERRAL, "false");
      idStoreCfg.setAttrMapping("firstname", "givenname");

      // Set operational config
      OperationalConfig opConfig = new OperationalConfig();
      opConfig.setEntityProperty(opConfig.USER_ENTITY, opConfig.SEARCH_BASE,
"cn=users,dc=us,dc=example,dc=com");
      opConfig.setEntityProperty(opConfig.USER_ENTITY, opConfig.CREATE_BASE,
"cn=users,dc=us,dc=example,dc=com");
      opConfig.setEntityProperty(opConfig.USER_ENTITY, opConfig.NAME_ATTR,
"cn");
      opConfig.setEntityProperty(opConfig.USER_ENTITY, opConfig.FILTER
_OBJCLASSES, "inetorgperson");
```

```
            opConfig.setEntityProperty(opConfig.USER_ENTITY, opConfig.CREATE
_OBJCLASSES, "inetorgperson");
            opConfig.setEntityProperty(opConfig.GROUP_ENTITY, opConfig.SEARCH_BASE,
 "cn=groups,dc=us,dc=example,dc=com");
            opConfig.setEntityProperty(opConfig.GROUP_ENTITY, opConfig.CREATE_BASE,
 "cn=groups,dc=us,dc=example,dc=com");
            opConfig.setEntityProperty(opConfig.GROUP_ENTITY, opConfig.NAME_ATTR,
 "cn");
            opConfig.setEntityProperty(opConfig.GROUP_ENTITY, opConfig.FILTER
_OBJCLASSES, "groupofuniquenames");
            opConfig.setEntityProperty(opConfig.GROUP_ENTITY, opConfig.CREATE
_OBJCLASSES, "groupofuniquenames");

        // Initialize Identity Store Service
        IdentityDirectoryFactory factory = new IdentityDirectoryFactory();
        ids = factory.getIdentityDirectory("ids1", entityCfg, idStoreCfg,
 opConfig);

        // Get UserManager and GroupManager handles
        uMgr = ids.getUserManager();
        gMgr = ids.getGroupManager();
    }

}
```

2.4.4 Add User

The following code sample adds a user to the identity store:

```
Principal principal = null;

        List<Attribute> attrs = new ArrayList<Attribute>();
        attrs.add(new Attribute("commonname", "test1_user1"));
        attrs.add(new Attribute("password", "mypassword".toCharArray()));
        attrs.add(new Attribute("firstname", "test1"));
        attrs.add(new Attribute("lastname", "user1"));
        attrs.add(new Attribute("mail", "test1.user1@example.com"));
        attrs.add(new Attribute("telephone", "1 650 123 0001"));
        attrs.add(new Attribute("title", "Senior Director"));
        attrs.add(new Attribute("uid", "tuser1"));

        try {
            CreateOptions createOpts = new CreateOptions();

            principal = uMgr.createUser(attrs, createOpts);



            System.out.println("Created user " + principal.getName());

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.5 Get User for Given Principal

The following code sample gets a user for a given principal.

```
User user = null;

        try {
            ReadOptions readOpts = new ReadOptions();

            user = uMgr.getUser(principal, readOpts);

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.6 Modify User

The following code sample modifies a user in the identity directory.

```
try {

            ModifyOptions modifyOpts = new ModifyOptions();

            List<ModAttribute> attrs = new ArrayList<ModAttribute>();
            attrs.add(new ModAttribute("description", "modified test user 1"));

            user.modify(attrs, modifyOpts);


            System.out.println("Modified user " + user.getName());
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.7 Get User for Given ID Value

The following code sample gets a user matching the given identity value.

```
try {

            ReadOptions readOpts = new ReadOptions();

            User user = uMgr.searchUser("tuser1", readOpts);

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.8 Search Users Using Complex Search Filter

The following code sample shows how to search for users using a complex search
filter.

```
try {

            // Complex search filter with nested AND and OR conditiions
            SearchFilter filter = new SearchFilter(
                SearchFilter.LogicalOp.OR,
                new SearchFilter(SearchFilter.LogicalOp.AND,
                  new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH,
 "test"),
                  new SearchFilter("telephone", SearchFilter.Operator.CONTAINS,
```

```
                "650")),
                        new SearchFilter(SearchFilter.LogicalOp.AND,
                          new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH,
                "demo"),
                          new SearchFilter(SearchFilter.LogicalOp.OR,
                            new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH,
                "hr"),
                            new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH,
                "it"),

                            new SearchFilter("telephone", SearchFilter.Operator.CONTAINS,
                "650")));

                // Requesting attributes
                List<String> reqAttrs = new ArrayList<String>();
                reqAttrs.add("jpegphoto");

                SearchOptions searchOpts = new SearchOptions();
                searchOpts.setPageSize(100);
                searchOpts.setRequestedAttrs(reqAttrs);
                searchOpts.setSortAttrs(new String[] {"firstname"});

                ResultSet<User> sr = uMgr.searchUsers(filter, searchOpts);
                while (sr.hasMore()) {
                    User user = sr.getNext();
                    System.out.println(user.getSubjectName());
                }

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.9 Change User Password

The following code sample shows how to change a user password.

```
ModifyOptions modOpts = new ModifyOptions();

        try {
            user.changePassword("welcome123".toCharArray(),
    "welcome1".toCharArray(), modOpts);
            System.out.println("Changed user password");
        } catch (Exception e) {
            System.out.println("Failed to change user password");
            e.printStackTrace();
        }
```

2.4.10 Reset User Password

The following code sample shows how to reset a user password.

```
ModifyOptions modOpts = new ModifyOptions();

        try {
            user.resetPassword("welcome123".toCharArray(), modOpts);
            System.out.println("Reset user password");
        } catch (Exception e) {
```

```
                    System.out.println("Failed to reset user password");
                    e.printStackTrace();
                }
```

2.4.11 Authenticate User

The following code sample shows how to authenticate a user.

```
          ReadOptions readOpts = new ReadOptions();
          try {
              User user = uMgr.authenticateUser("tuser1",
 "mypassword".toCharArray(), readOpts);
              System.out.println("authentication success");
          } catch (Exception e) {
              System.out.println("Authentication failed. " + e.getMessage());
              e.printStackTrace();
          }
```

2.4.12 Delete User

The following code sample shows how to delete a user.

```
try {
              DeleteOptions deleteOpts = new DeleteOptions();

              uMgr.deleteUser(principal, deleteOpts);

              System.out.println("Deleted user " + principal.getName());

          } catch (Exception e) {
              System.out.println(e.getMessage());
              e.printStackTrace();
          }
```

2.4.13 Create Group

The following code sample shows how to create a group.

```
Principal principal = null;

          List<Attribute> attrs = new ArrayList<Attribute>();
          attrs.add(new Attribute("name", "test1_group1"));
          attrs.add(new Attribute("description", "created test group 1"));
          attrs.add(new Attribute("displayname", "test1 group1"));
          try {
              CreateOptions createOpts = new CreateOptions();

              principal = gMgr.createGroup(attrs, createOpts);

              System.out.println("Created group " + principal.getName());
          } catch (Exception e) {
              System.out.println(e.getMessage());
              e.printStackTrace();
          }
```

2.4.14 Search Groups

The following code sample shows how to search groups.

```
public void searchGroups() {

        try {
            SearchFilter filter = new SearchFilter("name",
                                SearchFilter.Operator.BEGINS_WITH, "test");

            SearchOptions searchOpts = new SearchOptions();
            searchOpts.setPageSize(10);

            ResultSet<Group> sr = gMgr.searchGroups(filter, searchOpts);
            while (sr.hasMore()) {
                Group group = sr.getNext();
                System.out.println(group.getSubjectName());
            }

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.15 Get Management Chain

The following code sample shows how to get a management chain.

```
try {

            ReadOptions readOpts = new ReadOptions();
            User user = uMgr.searchUser("tuser1", readOpts);

            SearchOptions searchOpts = new SearchOptions();
            searchOpts.setPageSize(10);
            int nLevels = 0;

            ResultSet<User> sr  = user.getManagementChain(nLevels, searchOpts);
            while (sr.hasMore()) {
                User u = sr.getNext();
                System.out.println(u.getSubjectName());
            }

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
```

2.4.16 Get Reportees of a User

The following code sample shows how to get the reportees of user.

```
// Get Reportees with target search filter
    public void getReportees() {

        try {
            ReadOptions readOpts = new ReadOptions();
            User user = uMgr.searchUser("tuser1", readOpts);

            SearchOptions searchOpts = new SearchOptions();
```

```
                    searchOpts.setPageSize(20);
                    int nLevels = 0;

                    // get all the direct/indirect reporting of tuser1 who are
      "developers"
                    SearchFilter filter = new SearchFilter("title",
      SearchFilter.Operator.CONTAINS, "developer");
                    ResultSet<User> sr  = user.getReportees(nLevels, filter, searchOpts);
                    while (sr.hasMore()) {
                        User u = sr.getNext();
                        System.out.println(u.getSubjectName());
                    }

            } catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
```

2.4.17 Add a Member to a Group

The following code sample adds a member to a group.

```
try {
                    ReadOptions readOpts = new ReadOptions();
                    User user = uMgr.searchUser("tuser1", readOpts);
                    Group group = gMgr.searchGroup("test1_group1", readOpts);

                    ModifyOptions modOpts = new ModifyOptions();
                    user.addMemberOf(group, modOpts);

                    System.out.println("added tuser1 as a member of test1_group1");

            } catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
```

2.4.18 Delete a Member From Group

The following code sample deletes a member from a group.

```
try {
                    ReadOptions readOpts = new ReadOptions();
                    User user = uMgr.searchUser("tuser1", readOpts);
                    Group group = gMgr.searchGroup("test1_group1", readOpts);

                    ModifyOptions modOpts = new ModifyOptions();
                    group.deleteMember(user, modOpts);

                    System.out.println("deleted tuser1 from the group test1_group1");

            } catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
            }
```

2.4.19 Get All The Groups User is a Member

The following code sample gets all the groups in which user is a member.

```
try {
        ReadOptions readOpts = new ReadOptions();
        User user = uMgr.searchUser("tuser1", readOpts);

        SearchOptions searchOpts = new SearchOptions();
        searchOpts.setPageSize(10);
        int nLevels = 0;

        ResultSet<Group> sr  = user.getMemberOfGroups(nLevels, null,
searchOpts);
        while (sr.hasMore()) {
            Group group = sr.getNext();
            System.out.println(group.getSubjectName());
        }

    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
```

3

Using the ArisID API

This chapter describes the architecture and key functionality of the Identity Governance Framework ArisID API (ArisID API). The ArisID API provides enterprise developers and system architects a library for building identity-enabled applications using multiple identity protocols. The ArisID API enables developers to specify requirements for identity attributes, roles, and search filters by using Client Attribute Requirements Markup Language (CARML). This chapter contains the following topics:

3.1 About ArisID API

The Identity Governance Framework ArisID API represents a common core service through which all identity information exchanged should be passed. While not an official name, the ArisID API is often referred to as Identity Beans by developers.

The 11*g* Release 1 (11.1.1) release of the ArisID API is a subset of the configuration proposed at:

http://www.openliberty.org/wiki/index.php/ArisID_Configuration.

If you have installed Oracle WebLogic Server and Oracle Identity Management, all the necessary jar files for developing applications with this API are already installed on your computer.

> **See Also:** *Oracle Fusion Middleware Installation Guide for Oracle Identity Management* for information about installing Oracle Identity Management.

The Identity Governance Framework open source API jar files are as follows:

- **openliberty.arisId_1.1.jar** — Provides the core ArisID API with library functions and providers that can be used to retrieve identity subjects that contain collections of attributes. For more information, see http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/.

- **org.openliberty.arisIdBeans_1.1.jar** — Provides the ArisID beans, which provide Java object abstractions on top of the ArisID API. These convert the transactional approach of the ArisID API to an object or bean approach. For more information, see http://arisid.sourceforge.net/javadocs/arisId_1.1_javadoc/.

The ArisID API jar files are as follows:

- **idxuserrole.jar** — Provides the Standard User and Role identity read-only operations. This jar is generated from the standard idxuserrole.xml CARML file. For more information, see *Oracle Fusion Middleware Identity Governance Framework IDXUserRole Java API Reference*.

- **userrole.jar** — Provides the User and Role identity read/write operations for updating identity information. For more information, see *Oracle Fusion Middleware Identity Governance Framework UserRole Java API Reference*.

- **arisId-stack-ovd.jar** — This jar file is an implementation of the `IAttrSvcStack` interface with the Oracle Virtualization library to connect to different backends and provide an abstract view of the identity store entities.

The ArisID beans provide the Java APIs required for initialization and accessing CARML interactions. The bean generator generates a set of java files for each entity in the CARML file using Apache Velociy. The CARML file is a declarative document that describes the attribute usage requirements of your application. The ArisID beans are in the jar files idxuserrole.jar and userrole.jar.

The following figure provides a high-level view of the ArisID API architecture.

Figure 3–1   IGF ArisID API Architecture
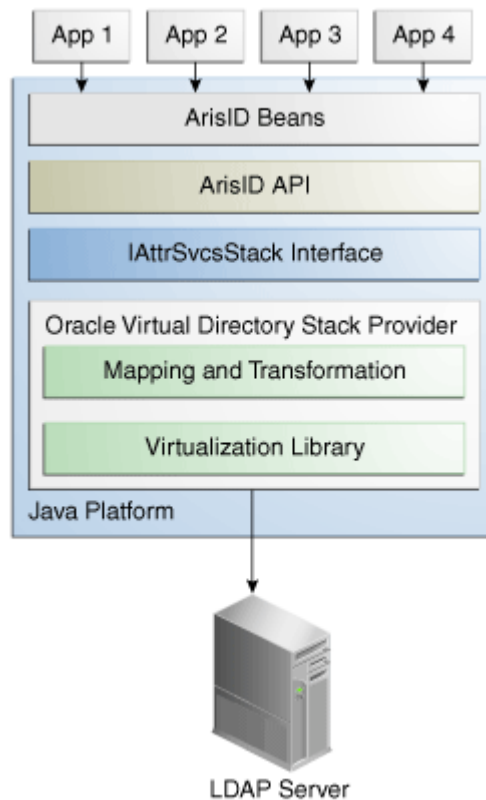
3.2 Configuring ArisID API

The Identity Governance Framework ArisID supports the basic development process Create > Modify > Test > Deploy. Creation requires CARML XML files and modifying them to suit your environment. Testing the application can be done in Oracle WebLogic Server embedded LDAP directory server.

This section contains the following topics:

- Configuring CARML Files
- Configuring the Identity Repository
- Configuring the Mapping File

3.2.1 Configuring CARML Files

Determine whether the existing ArisID beans meet your application's needs by examining the CARML files idxuserrole.xml (read-only operations) and userrole.xml (read-only and read/write operations). These files are located in DOMAIN_HOME/config/fmwconfig/carml.

3.2.2 Configuring the Identity Repository

The identity repository to be used by the ArisID beans must be available. You can use the Oracle WebLogic Server embedded LDAP-based directory server or any LDAP directory supported by 11*g* Oracle Virtual Directory. The ArisID API is integrated with Oracle Platform Security Services. It automatically connects to the LDAP-based identity store configured in Oracle Platform Security Services. For more information about the identity stores supported by Oracle Platform Security Services, see "System Requirements and Certification" on page 1-3.

For more information about Oracle Platform Security Services, see *Oracle Fusion Middleware Application Security Guide*.

If you must use a different identity store from the Oracle Platform Security Services identity store, then set the following system property:

```
igf.ovd.config.dir=DOMAIN_HOME/config/fmwconfig/arisidprovider/conf
```

Next, edit the adapters.os_xml file to include the `host`, `port` and credentials of the directory to be connected to. The `igf.ovd.config.dir` property can be set to any other directory containing adapaters.os_xml and other configuration files with the right settings.

Directory Limitations

The following LDAP directory limitations apply:

- When using OpenLDAP 2.2 or Novell eDirectory, there is no paging support. Paging APIs, for example, `searchUsersbyPage()`, `searchRolesbyPage()`, etc., do not work. There is no Multiple Language Support (MLS).

- When using Oracle WebLogic Server embedded LDAP-based directory server, there is no Multiple Language Support (MLS).

3.2.3 Configuring the Mapping File

When a CARML file is created a corresponding mapping file is created in the same location. The default mapping file has attribute details specific to Oracle WebLogic Server embedded directory server, which is the Oracle Platform Security Services

default identity store. If you are using a default CARML file and the Oracle Platform Security Services identity store, you do not need to configure mapping. The configuration parameters in Oracle Platform Security Services override the parameters in the mapping file.

3.3 Design Recommendations

The default CARML and mapping files make certain assumptions about the deployment scenario. You may need to modify these details depending on your deployment requirements. The configuration parameters that can be modified are discussed in this section.

This section contains the following topics:

- Choose a LoginID
- Choose a UniqueKey
- Specify Multiple Language Support
- Handle Large Results
- Secure the Application
- Configure the Timeout Interval

3.3.1 Choose a LoginID

In the default configuration, `email` is used as a unique identifier for identifying user entries. When you are searching for a user, the default attribute expected for search is `email`. For example:

```
SearchUser( String uniqueid,  Map<String, Object>)
```

For performance reasons, the attribute used as a unique identifier must be a searchable attribute in the backend. The mapping between the application's choice of `uniquekey` and the backend attribute is handled at configuration time. This is a configuration in Oracle Virtual Directory mapping. The `HashMap` is used to provide the optional context information to be used while performing the operation. In the current release it supports the following options:

- The Principal user that performs the search - (`ArisIdConstants.APP_CTX_AUTHUSER, (Principal)user`)
- The language constraint if any - (`ArisIdConstants.APP_CTX_LOCALE, "fr"`)
- Pagination support if any - (`ArisIdConstants.APP_CTX_PAGESIZE, 10`)

3.3.2 Choose a UniqueKey

An application occasionally stores the entries accessed from the identity repository's backend in their own application-specific repository. In such cases, you must carefully consider which attribute should be persisted. For instance, if the backend is an LDAP-based repository, you should use the `GUID` attribute as the persisting attribute because this is the only unique key on the LDAP-based backend. All other LDAP attributes are modifiable.

If the backend is a relational database, choose an attribute on which uniqueness constraint is enforced as the unique key. You can specify this in the ArisID mapping property file. The method to search for a user based on the unique key is:

```
searchUserOnUniqueKey(String UniqueKey, Map<String,Object>)
```

The `HashMap` is used to provide the optional context information to be used while performing the operation. In the current release it supports the following options:

- The Principal user that performs the search - (`ArisIdConstants.APP_CTX_ AUTHUSER, (Principal)user`)

- The language constraint if any - (`ArisIdConstants.APP_CTX_LOCALE, "fr"`)

- Pagination support if any - (`ArisIdConstants.APP_CTX_PAGESIZE, "10"`)

3.3.3 Specify Multiple Language Support

Multiple Language Support (MLS) is provided for applications that need locale-specific results. The attributes and the appropriate MLS code are stored in the ArisID properties file in the `multiLanguageAttributes` element.

```
<multiLanguageAttributes>…</multiLanguageAttribute>
```

Because `displayname` is the most commonly used multiple language attribute, it is configured by default as a multi-language attribute. Other attributes can be added as needed in the ArisID mapping file.

Restrictions

Any API to which locale is specified as an argument will return the locale-specific values for all the attributes listed in the ArisID properties file as `<multiLanguageAttributes>` that have locale-specific values. For all other attributes it returns the default values stored.

In the backend system, the data is returned in a form conforming to ISO-3166. For example, if there is a French locale (in addition to English), it is stored as `cn, :fr` for the `cn` attribute. The locale for the client applications should be specified in the properties `HashMap` as `ArisIdConstants.APP_CTX_LOCALE, "fr"` and the ArisID properties file should contain `cn` as `multiLanguageAttribute` and map this attribute.

3.3.4 Handle Large Results

When applications access identity data, the result set for a search is frequently too large to be handled by the application. In such cases you have the option of dividing the result into manageable sized pages. You do this by defining the number of objects to be returned in the page.

The following example shows a typical usage pattern:

```
RoleManager rm = new RoleManager(env);
  List<PropertyFilterValue> attrFilters = new  ArrayList<PropertyFilterValue>();
  attrFilters.add(new PropertyFilterValue(Role.NAME, "admin", AttributeFilter.OP_
CONTAINS));

  HashMap<String,Object> map = new HashMap<String,Object>();
  map.put("ArisIdConstants.APP_CTX_PAGESIZE","2");
  SearchResults<Role> sr = rm.searchRolesbyPage(attrFilters, map);

  while(sr.hasMore())
  {
    List<Role> roles = sr.getNextSet();

    for (int i=0; i<roles.size(); i++)
      //do the operations with roles.get(i)
}
```

3.3.5 Secure the Application

Two security scenarios are available for executing create, read, update, and delete (CRUD) operations on the target system. They are:

- Domain level credentials
- Application level credentials

Proxy authentication is not supported in this release.

3.3.5.1 Domain Level Credentials

In this scenario, all applications in a domain use common credentials to connect to the target system and perform operations with those credentials. The application does not maintain a footprint in the target system.

The LDAP Adapter's configuration file, `adapters.os_xml`, contains credentials to connect to the backend directory, along with the host and port details. If you do not provide any other credentials during initialization, the application connects to the target system using the credentials in the LDAP Adapter's configuration file.

If proxy user (logged in user id) is not specified in the API's application context, ArisID operation will be executed with the credentials that are in LDAP Adapter's configuration file.

If your application connects using common credentials, you must build security into the application itself so that it displays or modifies data only for an authorized user.

Example 3–1   Code Sample: adapters.os_xml

The LDAP Adapter's configuration file `adapters.os_xml` is configured with domain level userid and encrypted password to connect to backend directory. The following is a snippet of `adapters.os_xml`.

```
<binddn>cn=admin</binddn>
<bindpass>{OMASK}C2QXW1Nmf+s=</bindpass>
```

While initializing the ArisID API do not provide any credentials.

```
Map env = new HashMap();
// Do not set UserManager.SECURITY_PRINCIPAL & SECURITY_CREDENTIALS
UserManager uMgr  = new UserManager(env);
…
…
// Search Operation (with no proxy user in app context)
List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("User.FIRSTNAME", "app1",
AttributeFilter.OP_CONTAINS));
attrFilters.add(new PropertyFilterValue("User.LASTNAME", "user1",
AttributeFilter.OP_BGNSWITH));
Map<String, Object> appCtx = null;
users = um.searchUsers(attrFilters, appCtx);
```

3.3.5.2 Application Level Credentials

In this scenario, each application uses application level credentials to connect to the target system and performs CRUD operations with those credentials.

In this case you provide the application's user id and password while initializing the ArisID API. When you do that, the application connects to the target system using those credentials.

If no proxy user is specified in the API's application context then ArisID operation will be executed with the application's credentials.

This scenario has the following features:

- Each application has different privileges to view and update the data in the target system

- You can audit the modifications performed by each application in the target system

Example 3–2   Code Sample: adapters.os_xml

The LDAP Adapter's configuration file `adapters.os_xml` is configured with domain level userid and encrypted password to connect to backend directory. The following is a snippet of `adapters.os_xml`.

```
<binddn>cn=admin</binddn>
    <bindpass>{OMASK}C2QXW1Nmf+s=</bindpass>
```

While initializing the ArisID API, provide the application user credentials.

```
Map env = new HashMap();
env.put(UserManager.SECURITY_PRINCIPAL, "cn=app1_
user,cn=users,dc=example,dc=com");
env.put(UserManager.SECURITY_CREDENTIALS, "mypassword");
UserManager uMgr  = new UserManager(env);
…
// Search Operation (with no proxy user in app context)
List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("User.FIRSTNAME", "app1",
AttributeFilter.OP_CONTAINS));
attrFilters.add(new PropertyFilterValue("User.LASTNAME", "user1",
AttributeFilter.OP_BGNSWITH));
Map<String, Object> appCtx = null;
users = um.searchUsers(attrFilters, appCtx);
```

3.3.6 Configure the Timeout Interval

The default connect/read timout is configured to be 15 seconds. For example, if the LDAP operation on `IdentityStore` is taking more than 15 seconds, the operation will timeout and throw the following IGF exception:

```
org.openliberty.arisid.stack.ConnectionException
```

If the `IdentityStore` has a many entries and the applications are using filters with "contains" and search with paging/sorting, those queries may timeout.

The recommendation is to set the timeout value to 0 (no timeout) and increase the pool size to 20. If the preference is for the application has some timeout interval, set the value to greater than 0.

To configure the timeout interval:

1. Run the following WLST command to list all adapters.

   ```
   listAdapters()
   ```

2. Run the following command for each adapter to set `timeout` and `maxpoolsize`.

   a. `modifyLDAPAdapter('<ADAPTER NAME>', 'OperationTimeout', 0)`

   b. `modifyLDAPAdapter('<ADAPTER NAME>', 'MaxPoolSize', 20)`

3. Restart WebLogic Server.

3.4 Generating ArisID Beans

Generate ArisID beans as follows:

```
java BeanGenerator [-genmap] <package name> <output dir> [<relationship file>]
<carml file>
```

where:

- -genmap: Generates the CARML-ArisIDStack map file in the same location where the CARML file is.

- package name: Is the package name for the generated bean classes.

- output dir: Location to write the generated bean classes.

- relationship file: File containing the relationship definitions between entities.

- carml file: The CARML declaration file.

To build ORG beans from a CARML file:

1. Create a CARML file name org.xml with the appropriate attributes/interactions for ORG entity.

2. Generate org beans (OrgManager.java and Org.java) using the Bean Generator. The build.xml file should resemble the following sample:

```
<path id="ArisIDBeans.classpath">
            <pathelement location="MW_HOME/oracle_
common/modules/velocity-dep-1.4.jar" />
            <pathelement location="MW_HOME/oracle_common/modules/oracle.jrf_
11.1.1/jrf.jar" />
    </path>
    <property name="BeanGeneratorClassPath" refid="ArisIDBeans.classpath"/>
    <target name="generatebeans" description="generate arisid beans">
        <java classname="org.openliberty.arisidbeans.BeanGenerator"
dir="${generatedsource.dir}" fork="true">
            <arg value="${generatedbean.userrole.packagename}"/>
            <arg value="."/>
            <arg value="${carml.dir}/org.xml"/>
            <classpath>
                <pathelement path="${BeanGeneratorClassPath}"/>
            </classpath>
            <sysproperty key="org.openliberty.arisid.policy.wspolicy.class"

value="org.openliberty.arisid.policy.neethi.PolicyImpl" />
        </java>
    </target>
```

3. Compile the generated java files Org.java and OrgManager.java.

4. Edit the generated mapping file (igf-map-config-.xml) to update basesearch, objectclass, OVD attribute names with their values.

5. An application can use the generated ORG APIs for the interactions defined in the CARML file (org.xml). After the application is deployed in application server:

   a. Copy the mapping file under DOMAIN_HOME/config/fmwconfig/arisidprovider/conf, and

   b. Copy the CARML file under DOMAIN_HOME/config/fmwconfig/carml

3.5 Sample Application Using IDX User and Role Beans

The following sample application uses IDX User/Role Beans.

3.5.1 SearchUsers.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@page import="org.openliberty.arisid.*"%>
<%@page import="org.openliberty.arisidbeans.*"%>
<%@page import="oracle.igf.userrole.*"%>
<%@page import="java.util.*"%>
<%@page import="java.net.URI"%>
<%!public static UserManager uMgr = null;
{
        try {
                uMgr = new UserManager(null);
        } catch (Exception e) {
                e.printStackTrace();
        }

}
%>
<html>
<head>
<title>Search Users</title>
<%


String firstname = request.getParameter("firstname");
String lastname = request.getParameter("lastname");
String telephone = request.getParameter("telephone");


List<PropertyFilterValue> attrFilters = new ArrayList<PropertyFilterValue>();
attrFilters.add(new PropertyFilterValue("firstname", firstname,
AttributeFilter.OP_BGNSWITH));
attrFilters.add(new PropertyFilterValue("lastname", lastname, AttributeFilter.OP_
BGNSWITH));
attrFilters.add(new PropertyFilterValue("telephone", telephone,
AttributeFilter.OP_CONTAINS));

List<User> subjs = uMgr.searchUsers(attrFilters);

%>
</head>
<body>

<a href="SearchUsers.html">Home</a>
<center>List of Users with FirstName starting with "<%=firstname%>", LastName
starting with "<%=lastname%>" and TelephoneNumber containing
"<%=telephone%>"</center>

<%
Iterator<User> sIter = subjs.iterator();
while (sIter.hasNext()) {
        User subj = sIter.next();
```

```
                Map<String, IAttributeValue> vals = subj.getAllAttributes();
                Iterator<IAttributeValue> iter = vals.values().iterator();
%>
<table border="0">
        <tr>
                <th>Item</th>
                <th>Value</th>
        </tr>
        <%
                while (iter.hasNext()) {
                        IAttributeValue val = iter.next();
                        String name = val.getNameIdRef();
                        String value = null;
                        if (val.size() > 0)
                                value = val.get(0);
if (value != null)
{
        %>
        <tr>
                <td><%=name%></td>
                <td><%=value%></td>
        </tr>
        <%
}
                }
        %>
</table>
<%
        }
%>
<br>
<br>
<br>
<a href="SearchUsers.html">Home</a>
</body>
</html>
```

3.5.2 SearchUsers.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD><TITLE>Search Users</TITLE></HEAD>
<BODY>
<FORM METHOD=POST ACTION="SearchUsers.jsp">

First Name Starting with <INPUT TYPE=TEXT NAME=firstname SIZE=30><BR><BR>
Last Name Starting with <INPUT TYPE=TEXT NAME=lastname SIZE=30><BR><BR>
Telephone Number containing <INPUT TYPE=TEXT NAME=telephone SIZE=15><BR><BR>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

3.6 OpenLDAP Considerations

For OpenLDAP, Role.MEMBER is a mandatory attribute for the following APIs:

■   createRole(List<PropertyValue> attrVals, Map<String,Object> appCtxMap)

- `createRole(List<PropertyValue> attrVals)`

If the `Role.MEMBER` is not included in the input `attrVals` list, role creation will fail.

4

Migrating to Identity Directory API

This chapter describes how to migrate applications from the User and Role API to the Identity Directory API. This chapter contains the following topics:

- Section 4.1, "Introduction"
- Section 4.2, "Migrate the Application"
- Section 4.3, "Comparison Between User and Role API and IDS API"

4.1 Introduction

If you have an application that uses the User and Role API described in *Oracle Fusion Middleware Application Security Guide* and *Oracle Fusion Middleware User and Role Java API Reference for Oracle Platform Security Services*, you can modify it to use Identity Directory API instead.

The Identity Directory API also picks up the LDAP-based identity store confirmation from the jps-config file. As such, when migrating an application from the User and Role API to the Identity Directory API you do not need to change the configuration in the jps-config file.

Applications that initialize the User and Role API with a programmatic configuration can use a similar method to initialize the Identity Directory API. For more information, see Section 2.4.3, "Initialize and Obtain In-Memory Identity Directory Handle".

4.2 Migrate the Application

Application migrating from the User and Role API to the Identity Directory API need to make the following code changes:

- API Initialization
- Getting UserManager and GroupManager Handle
- Search Filter
- CRUD Operations
- User-Related APIs
- Role-Related APIs

4.2.1 API Initialization

Process is similar to using `IdentityStoreService.GetIdmStore()` for getting `oracle.security.idm.IdentityStore` handle. Identity Directory API uses

IdentityStoreService.getIdentityStore() to get IdentityDirectory handle. For example:

```
import oracle.igf.ids.IdentityDirectory;
import oracle.igf.ids.IDSException;
import oracle.security.jps.JpsContext;
import oracle.security.jps.JpsContextFactory;
import oracle.security.jps.service.idstore.IdentityStoreService;

// Get IdentityDirectory from JpsContext
JpsContext context = JpsContextFactory.getContextFactory().getContext();
IdentityStoreService idstore = (IdentityStoreService)
context.getServiceInstance(IdentityStoreService.class);
Identity Directory ids = idstore.getIdentityStore();
```

4.2.2 Getting UserManager and GroupManager Handle

User related CRUD operations can be performed with oracle.igf.ids.UserManager and Role related CRUD operations can be performed with oracle.igf.ids.GroupManager. UserManager and GroupManager handles can be obtained from IdentityDirectory object. For example:

```
import oracle.igf.ids.UserManager;
import oracle.igf.ids.GroupManager;

// Get UserManager and GroupManager handles
        UserManager uMgr = ids.getUserManager();
        GroupManager gMgr = ids.getGroupManager();
```

4.2.3 Search Filter

A simple or complex search filter can be built using oracle.igf.ids.SearchFilter. For example:

```
import oracle.igf.ids.SearchFilter;

// Simple search filter for (firstname equals "john")

SearchFilter filter1 = new SearchFilter("firstname",
SearchFilter.Operator.EQUALS, "john");

// Complex search filter for
   ((title contains "manager") and (org equals "amer")) or
((title contains "senior manager") and (org equals "apac"))

            SearchFilter filter = new SearchFilter(
                SearchFilter.LogicalOp.OR,
                new SearchFilter(SearchFilter.LogicalOp.AND,
                  new SearchFilter("manager", SearchFilter.Operator.CONTAINS,
 "manager"),
                  new SearchFilter("org", SearchFilter.Operator.EQUALS, "amer")),
                new SearchFilter(SearchFilter.LogicalOp.AND,
                  new SearchFilter("manager", SearchFilter.Operator.CONTAINS,
 "senior manager"),
                  new SearchFilter("org", SearchFilter.Operator.EQUALS, "apac")));
```

4.2.4 CRUD Operations

Create/Read/Update/Delete (CRUD) operations on User, Group, Org, and generic entities are discussed in the following sections:

-
-
-
-
-
-

4.2.4.1 Finding a User

The following APIs are used for finding a user:

- Get user for given `principal` identifier. For example:

  ```
  User getUser(Principal principal, ReadOptions opts)
  ```

- Search for user matching given `id` attribute value that uniquely identifies the user. For example:

  ```
  User searchUser(String id, ReadOptions opts)
  ```

- Finds user matching given attribute name and value. For example:

  ```
  User searchUser(String attrName, String attrVal, ReadOptions opts)
  ```

- Search for user matching given GUID value that uniquely identifies the user. For example:

  ```
  searchUserByGuid(String guid, ReadOptions opts)
  ```

4.2.4.2 Searching Users

The following is an example of the API for searching a user.

```
ResultSet<User> searchUsers(SearchFilter filter, SearchOptions opts)
```

4.2.4.3 Creating a User

The following is an example of the API for creating a user.

```
Principal createUser(List<Attribute> attrVals, CreateOptions opts)
```

4.2.4.4 Deleting a User

The following are examples of the API for deleting a user.

- Delete the user given the `principal` identifier.

  ```
  void deleteUser(Principal principal, DeleteOptions opts)
  ```

- Delete the user given the `id` attribute value.

  ```
  void deleteUser(String id, DeleteOptions opts)
  ```

4.2.4.5 Authenticating a User

The following are examples of the API for user authentication.

- Authenticate the user matching the given `id` attribute value.

  ```
  User authenticateUser(String id, char[] password, ReadOptions opts)
  ```

- Authenticate the user for given `principal` identifier.

  ```
  boolean authenticateUser(Principal principal, char[] password)
  ```

4.2.4.6 Modifying Users and Managing Related Entities

The APIs for modifying user attributes and for getting the related entities are in `User` object instead of `UserManager`.

4.2.4.6.1 Modifying a User The following are examples of the API for modifying a user.

- Modify user attributes.

  ```
  void User.modify(List<ModAttribute> attrVals, ModifyOptions opts)
  ```

- Set the user attribute value.

  ```
  void User.setAttributeValue(String attrName, String attrVal, ModifyOptions
  opts)
  ```

4.2.4.6.2 Managing Related Entities The following are examples of the APIs for managing entities.

- Get the management chain.

  ```
  ResultSet<User> getManagementChain(int nLevels, SearchOptions opts)
  ```

- Check if the given user is manager of this user.

  ```
  boolean isManager(User user, boolean direct, ReadOptions opts)
  ```

- Set the given user as manager of this user.

  ```
  void setManager(User user, ModifyOptions opts)
  ```

- Get all the reportees of this user.

  ```
  ResultSet<User> getReportees(int nLevels,
   SearchFilter targetFilter, SearchOptions opts)
  ```

- Get all the groups this user is a member of and matching the given filter criteria.

  ```
  ResultSet<Group> getMemberOfGroups(int
   nLevels, SearchFilter targetFilter, SearchOptions opts)
  ```

- Check if this user is a member of the given group.

  ```
  boolean isMemberOf(Group group, boolean direct, ReadOptions opts)
  ```

- Add this user as a member to given group.

  ```
  void addMemberOf(Group group, ModifyOptions opts)
  ```

- Delete this user as a member to given group.

  ```
  void deleteMemberOf(Group group, ModifyOptions opts)
  ```

4.3 Comparison Between User and Role API and IDS API

The differences between the User and Role API and Identity Directory API are discussed in the following topics:

- User-Related APIs

- Role-Related APIs

4.3.1 User-Related APIs

Table 4–1 provides a comparison between the User-related API method and the corresponding Identity Directory API API method.

Table 4–1   Comparison Between User-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
User Creation	User UserManager.createUser(String name, char[] password)	Principal UserManager.createUser(List<Attribute> attrVals, CreateOptions opts)
	User UserManager.createUser(String name, char[] password, PropertySet pset)	
Delete User	void UserManager.dropUser(UserProfile user)	void UserManager.deleteUser(Principal principal, DeleteOptions opts)
	void UserManager.dropUser(User user);	void UserManager.deleteUser(String id, DeleteOptions opts)
Authenticate User	User UserManager.authenticateUser(String user_id, char[] passwd)	User UserManager.authenticateUser(String id, char[] password, ReadOptions opts)
	User UserManager.authenticateUser(User user, char[] passwd)	boolean UserManager.authenticateUser(Principal principal, char[] password)
	User UserManager.authenticateUser(String user_id, String authProperty, char[] passwd)	
Check if create User is supported	boolean UserManager.isCreateUserSupported()	boolean UserManager.getCapabilities().isCreateCapable()
Check if modify User is supported	boolean UserManager.isModifyUserSupported()	boolean UserManager.getCapabilities().isUpdateCapable()
Check if drop User is supported	boolean UserManager.isDropUserSupported()	boolean UserManager.getCapabilities().isDeleteCapable()
Search Users by given search criteria	SearchResponse IdentityStore.searchUsers(SearchParameters params)	ResultSet<User> UserManager.searchUsers(SearchFilter filter, SearchOptions opts)

Table 4–1   (Cont.) Comparison Between User-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
Search an User by name/uniquename /guid	User IdentityStore.searchUser(String name)	User UserManager.searchUser(String id, ReadOptions opts)
		User UserManager.searchUser(String attrName, String attrVal, ReadOptions opts)
Check if User exists in the repository for a given User object	boolean IdentityStore.exists (User user)	User.getPrincipal() if the following method returns null user doesn't exist; otherwise exists
		User getUser(Principal principal, ReadOptions opts)
Simple search filter (search based on a single attribute name, type and value)	SimpleSearchFilter	SearchFilter(String propertyName, Operator op, String propertyVal)
Complex Search Filter (search based on more than one attribute with filter conditions and nested filters)	ComplextSearchFilter	SearchFilter(LogicalOp op, SearchFilter... searchFilters)
Getting a property value for a given property name	String User.getPropertyVal(String propName) (User Role API fetches the attribute values from cache. If it misses cache, it fetches from repository)	String User.getAttributeValue(String attrName) Limitation: Returns attribute values from User object that has been already fetched from the repository.
Getting the User property for a given property name	Property User.getProperty(String propName)	Attribute User.getAttribute(String attrName)
Getting the user properties for a given set of property names	Map User.getProperties()	Map<String, Attribute> User.getAllAttributes()
Get all user properties from the repository for a user	PropertySet User.getAllUserProperties()	Map<String, Attribute> User.getAllAttributes()
Get all user property names from the schema	List IdentityStore.getUserProper yNames() Returns the names of all the properties in the schema	List<String> UserManager.getEntityAttributes()
Changing the attribute value in the repository of an user	void User.setProperty(ModProper ty mprop)	void User.setAttributeValue(String attrName, String attrVal, ModifyOptions opts)

Table 4–1   (Cont.) Comparison Between User-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
Changing the set of attribute values in the repository for an user	void User.setProperties(ModProperty[] modPropObjs) void User.setProperties(LdapContext ctx, ModProperty[] modPropObjs)	void User.modify(List<ModAttribute> attrVals, ModifyOptions opts)
Get all the reportees of an User either direct or indirect	SearchResponse User.getReportees(boolean direct)	ResultSet<User> User.getReportees(int nLevels, SearchFilter targetFilter, SearchOptions opts)
Get Management chain of an user	List User.getManagementChain(int max, String upToManagerName, String upToTitle)	ResultSet<User> User.getManagementChain(int nLevels, SearchOptions opts) List<User> User.getManagementChain(int nLevels, String manager, String title, SearchOptions opts)
Get/Set of Binary Attributes	Available Property in User/Role API supports binary attributes byte[] user.getJPEGPhoto() void user.setJPEGPhoto(String imgpath)	Returns base64 encoded value While setting the value either base64 encoded value or byte[] can be used for creating ModAttribute.
Selecting the Realm	Available env.put(OIDIdentityStoreFactory.RT_SUBSCRIBER_ NAME, "<realm dn>"); IdentityStoreFactory.getIdentityStoreInstance(env);	This is part of IDS Operational configuration. At API level searchbase and createbase can be specified as well.

4.3.2 Role-Related APIs

Table 4–2 provides a comparison between the Role-related API method and the corresponding Identity Directory API method.

Table 4–2   Comparison Between Role-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
Creating a Role	Role RoleManager.createRole(String name, int scope) Role RoleManager.createRole(String name)	Principal GroupManager.createGroup(List<Attribute> attrVals, CreateOptions opts)
Deleting a Role	void RoleManager.dropRole(Role Profile role) void RoleManager.dropRole(Role role)	void GroupManager.deleteGroup(Principal principal, DeleteOptions opts)

Table 4–2 (Cont.) Comparison Between Role-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
Check if create role is supported	boolean RoleManager.isCreateRoleSupported()	boolean GroupManager.getCapabilities().isCreateCapable()
Check if modify role is supported	boolean RoleManager.isModifyRoleSupported()	boolean GroupManager.getCapabilities().isUpdateCapable()
Check if delete role is supported	boolean RoleManager.isDropRoleSupported()	boolean GroupManager.getCapabilities().isDeleteCapable()
Is the Group owned by a User	boolean RoleManager.isGranted(Role parent, Principal principal)	boolean Group.isMember(User user, boolean direct, ReadOptions opts) boolean User.isMemberOf(Group group, boolean direct, ReadOptions opts)
Is the Group owned by a User	boolean RoleManager.isOwnedBy(Role parent, Principal principal)	boolean User.isOwnerOf(Group group, boolean direct, ReadOptions opts)
Is the group managed by a User	boolean RoleManager.isManagedBy(Role parent, Principal principal)	Not supported
Get all the members of a Role either direct / indirect	SearchResponse Role.getGrantees(SearchFilter filter, boolean direct)	ResultSet<User> Group.getMembers(int nLevels, SearchFilter targetFilter, SearchOptions opts)
Add an user as a member to a role	void RoleManager.grantRole(Role parent, Principal principal)	void Group.addMember(User user, ModifyOptions opts)
Remove a user from being member of a role	void RoleManager.revokeRole(Role parent, Principal principal)	void Group.deleteMember(User user, ModifyOptions opts)
Get all the owners of a specific Role either direct / indirect	SearchResponse Role.getOwners(SearchFilter filter, boolean direct) SearchResponse Role.getOwners(SearchFilter filter)	ResultSet<User> Group.getOwners(int nLevels, SearchFilter targetFilter, SearchOptions opts)
Add a user as a owner of a role	void Role.addOwner(Principal principal)	void Group.addOwner(User user, ModifyOptions opts)
Remove a user from being a owner of a Role	void Role.removeOwner(Principal principal)	void Group.deleteOwner(User user, ModifyOptions opts)
Get all the managers of a Role either direct / indirect	SearchResponse Role.getManagers(SearchFilter filter, boolean direct) SearchResponse Role.getManagers(SearchFilter filter)	Not Supported
Add a user as a manager of a Role	void Role.addManager(Principal principal)	Not Supported

Table 4–2 (Cont.) Comparison Between Role-Related API and Identity Directory API

Functionality	User/Role API Method	Identity Directory API Method
Remove a user from being manager of a Role	void Role.removeManager(Principal principal)	Not Supported
Getting the role property	Property Role.getProperty(String propName) Note: User Role API fetches these attribute values from cache. If it misses cache, it fetches from repository.	Attribute Group.getAttribute(String attrName)
Determine the Role Type	Role.isApplicationRole Role.isEnterpriseRole Role.isSeeded	Not Supported
Search Roles for a given search criteria	SearchResponse IdentityStore.searchRoles(int scope, SearchParameters params)	ResultSet<Group> GroupManager.searchGroups(SearchFilter filter, SearchOptions opts)
Search a Role by name/uniquename /guid	Role IdentityStore.searchRole(int searchType, String value)	Group searchGroup(String id, ReadOptions opts) Group searchGroup(String attrName, String attrVal, ReadOptions opts)
Search both User and Roles for a given filter	SearchResponse IdentityStore.search(SearchParameters params)	Available through separate methods: UserManager.searchUsers GroupManager.searchGroups
Get all the roles assigned to user/group	SearchResponse getGrantedRoles(Principal principal, boolean direct)	ResultSet<Group> User.getMemberOfGroups(int nLevels, SearchFilter targetFilter, SearchOptions opts) ResultSet<Group> Group.getMemberOfGroups(int nLevels, SearchFilter targetFilter, SearchOptions opts)
Get all the roles owned by user/group	SearchResponse getOwnedRoles(Principal principal, boolean direct)	ResultSet<Group> User.getOwnedGroups(int nLevels, SearchFilter targetFilter, SearchOptions opts) ResultSet<Group> Group.getOwnedGroups(int nLevels, SearchFilter targetFilter, SearchOptions opts)
Get all the roles managed by user/group	SearchResponse getManagedRoles(Principal principal, boolean direct)	Not supported