Oracle® Fusion Middleware

Developing Applications with Identity Governance Framework

12.2.1.2

E78161-01

October 2016

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 Fusion Middleware Developing Applications with Identity Governance Framework, 12.2.1.2

E78161-01

Primary Author: Devanshi Mohan

Contributors: Santhosh H M, Anandaprakash Sivasubramanian

Contents

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 that have purchased support 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:

- *Java API Reference for Identity Governance Framework Identity Directory*
- *Java API Reference for Identity Governance Framework IDXUserRole*
- *Java API Reference for Identity Governance Framework UserRole*
- *Securing Applications with Oracle Platform Security Services*
- *Oracle® Fusion Middleware Infrastructure Security WLST Command Reference*
- 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 in This Guide

Review the new features and significant product changes for the Identity Governance Framework (IGF) and the related developer APIs.

New Features in Release 12*c* (12.2.1.2)

This revision contains no new features. Minor updates were made throughout the guide.

New Features in Release 12*c* (12.2.1.1)

The new features and major changes introduced in release 12.2.1.1 are as follows:

- New IDS Tuning Configuration Parameters

New IDS Tuning Configuration Parameters

New configuration parameter `Protocols` has been added for tuning your IDS deployment.

See Configuration Parameters for IDS.

New Features in Release 12*c* (12.2.1)

The new features and major changes introduced in release 12.2.1.0.0 are as follows:

- New IDS Configuration Parameters
- Support to Reclaim Connection Back into the Connection Pool

New IDS Configuration Parameters

New configuration parameters `MaxPoolConnectionReuseTime` and `ConnectTimeout` have been added for tuning your IDS deployment.

See Configuration Parameters for IDS.

Support to Reclaim Connection Back into the Connection Pool

You can now configure connection pool property, which allows you to reclaim borrowed connections into the connection pool after a connection has not been used for a specific time duration.

Use the `PoolConnectionReclaimTime` configuration parameter to set the property. This feature helps maximize connection reuse and conserves system resources that are otherwise lost on maintaining borrowed connections that are no longer in use.

See Configuration Parameters for IDS.

New Features in Release 12*c* (12.1.3)

The new features and major changes introduced in release 12.1.3 are as follows:

- Logical NOT Operator in Search Filters
- Pass-Through Support for Entity Attributes

Logical NOT Operator in Search Filters

You can specify the logical NOT operator to negate a simple or complex filter condition.

The `SearchFilter` class has two new methods: `negate()` is a toggle method to set the NOT condition, and the `isNegated()` method can check if the NOT condition is already set.

See Using Logical NOT Operator in a Search Filter.

Pass-Through Support for Entity Attributes

The Identity Directory API supports entity attribute pass-through.

You do not need to include each and every attribute in attribute definitions and attribute references under the entity definition. You can include any attribute in an add, modify, requested attributes, or search filter operation, and if the attribute is not present in the in the identity store schema, the IDS API returns the error thrown by the identity store.

See Logical Entity Configuration for an Identity Directory Service.

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.

The following topics provide an introduction to the Identity Governance Framework and the related developer APIs Oracle has made available:

- Overview of the Identity Governance Framework
- Understanding Identity Governance Framework APIs
- System Requirements and Certification for Identity Governance Framework

1.1 Overview of the Identity Governance Framework

The Identity Governance Framework enables enterprises to define standards that secures the exchange of identity information and regulates compliance between applications both internally and with the external world. Identity information may include data such as names, addresses, numbers, and other information associated with an individual's identity.

The Identity Governance Framework is an open initiative. As part of this initiative, Oracle has contributed key initial specifications and is making them available to the community.

The Identity Governance Framework is 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.

The 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 of Identity Governance Framework to Organizations
- Benefits of Identity Governance Framework to Developers

1.1.1 Benefits of Identity Governance Framework to Organizations

The Identity Governance Framework makes use of the policies and standards that helps support enterprise security and provides an assurance to the users that the identity information is secured and managed appropriately by the parties to whom it has been entrusted.

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 of Identity Governance Framework 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 standard approach to write applications that use this data so that governing policies can be used to control it. This results 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 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 Using the ArisID API .

1.2 Understanding Identity Governance Framework APIs

The Identity Governance Framework depends on two specific API modules that enables the organizations to implement the data model requirements required to ensure security and compliance while exchange of identity information.

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 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 Using the ArisID API .

1.3 System Requirements and Certification for Identity Governance Framework

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.

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).

For more information, see *Oracle Fusion Middleware System Requirements and Specifications*.

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products. For more information, see *Oracle Fusion Middleware Supported System Configurations*.

2 Using the Identity Directory API

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.

The following topics describe the architecture and key functionality of the Identity Governance Framework, Identity Directory API, and Identity Directory Service.

- Overview of the Identity Directory API
- Identity Directory API Configuration
- Design Recommendations for Identity Directory API
- Examples of Using the Identity Directory API

2.1 Overview of the 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 can be fully configured 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 for identity information regulation and control. The API can be used in both Java EE and Java SE modes. For more information about the Identity Governance Framework, see 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 Unified Directory, Oracle Directory Server EE, and Active Directory.

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.

This section contains the following topics:

- Understanding Identity Directory API
- Identity Directory Service Architecture

2.1.1 Understanding Identity Directory API

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

For more information about the Identity Directory Service, also referred to as the Identity Store Service, see “Introduction to the Identity Store Service” in *Oracle® Fusion Middleware Securing Applications with Oracle Platform Security Services*.

2.1.2 Identity Directory Service Architecture

To use the Identity Directory Service APIs, it is essential to understand the architecture to learn how the identities are integrated, and how they can be used.

Figure 2-1 shows the logical architecture of the Identity Directory Service.

Figure 2-1 Identity Directory API Architecture

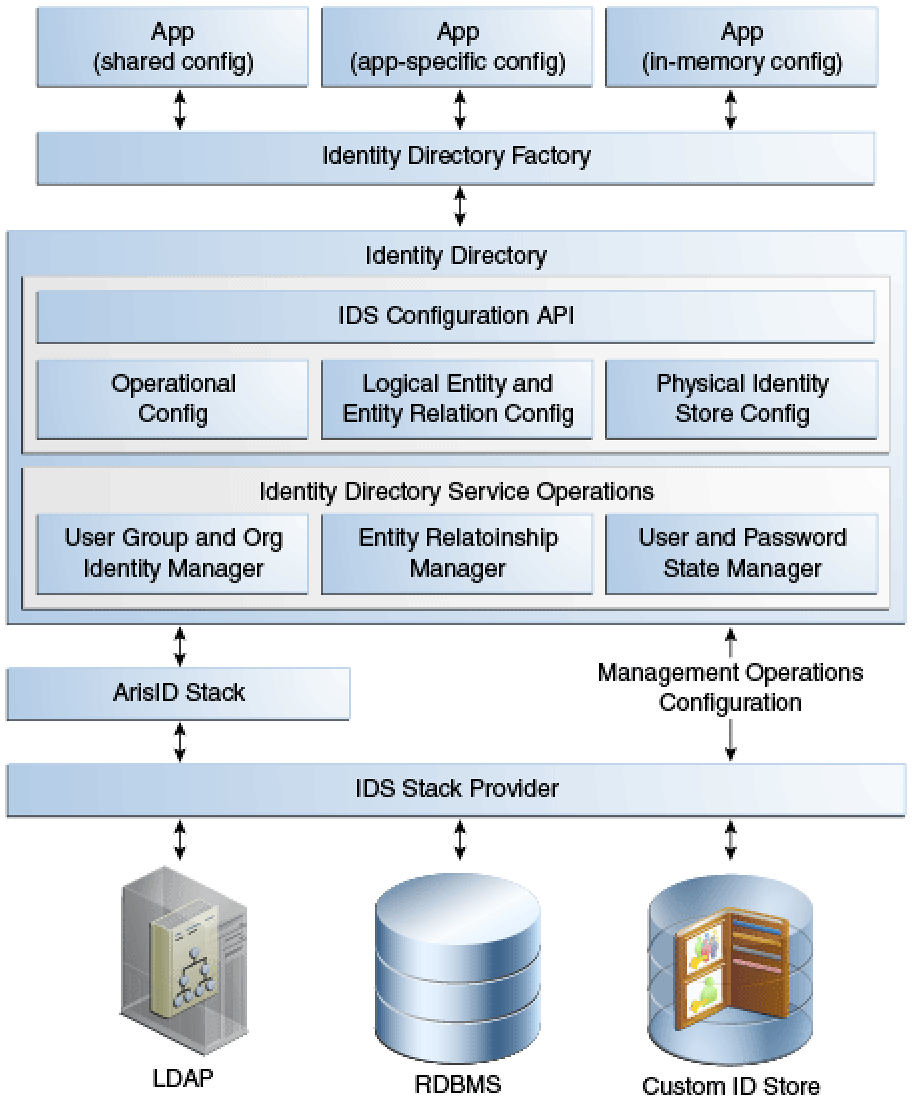
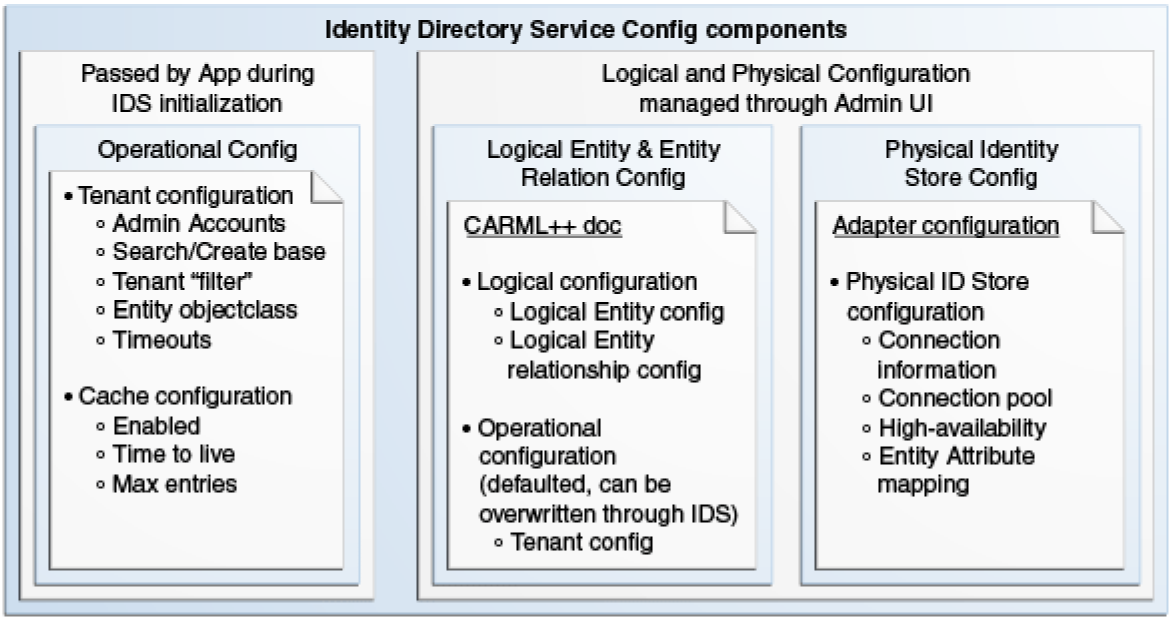

App
(shared config)
App
(app-specific config)
App
(in-memory config)
Identity Directory Factory
Identity Directory
IDS Configuration API
Operational Config
Logical Entity and Entity Relation Config
Physical Identity Store Config
Identity Directory Service Operations
User Group and Org Identity Manager
Entity Relatoinship Manager
User and Password State Manager
ArisID Stack
Management Operations Configuration
IDS Stack Provider
LDAP
RDBMS
Custom ID Store

Figure 2-2 shows the relationship between the Identity Directory Service components.

Figure 2-2 Identity Directory API Components


Identity Directory Service Config components
Passed by App during IDS initialization
Operational Config
• Tenant configuration
◦ Admin Accounts
◦ Search/Create base
◦ Tenant "filter"
◦ Entity objectclass
◦ Timeouts
• Cache configuration
◦ Enabled
◦ Time to live
◦ Max entries
Logical and Physical Configuration managed through Admin UI
Logical Entity & Entity Relation Config
CARML++ doc
• Logical configuration
◦ Logical Entity config
◦ Logical Entity relationship config
• Operational configuration (defaulted, can be overwritten through IDS)
◦ Tenant config
Physical Identity Store Config
Adapter configuration
• Physical ID Store configuration
◦ Connection information
◦ Connection pool
◦ High-availability
◦ Entity Attribute mapping

2.2 Identity Directory API Configuration

The Identify Directory API provides an interface to access and modify users and group information from different identity stores. It is a combination of the logical entity configuration, the physical identity store configuration, and the 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 for an Identity Directory Service
- Physical Identity Store Configuration for an Identity Directory Service
- Operational Configuration for an Identity Directory Service

2.2.1 Logical Entity Configuration for an Identity Directory Service

It is important to maintain and control the attributes and properties that are associated with a logical entity configuration for an Identity Directory.

The following topics describe the logical entity configuration information for an Identity Directory Service:

- Properties of a Logical Entity Configuration
- Attributes of a Logical Entity Configuration

- Properties of a Logical Entity Definition
- Properties of a Logical Entity Relationship

2.2.1.1 Properties of a Logical Entity Configuration

You must keep in mind the properties of a logical entity configuration.

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.

2.2.1.2 Attributes of a Logical Entity Configuration

The following table describes the 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.

Note:

Beginning with the 12*c* (12.1.3) release, the Identity Directory API supports entity attribute pass-through. With pass-through support, you do not need to include each and every attribute in attribute definitions (described in table in Attributes of a Logical Entity Configuration) and in attribute references under the entity definition (described in table in Properties of a Logical Entity Definition).

The IDS API allows any attribute in an add, modify, requested attributes, or search filter operation. The entity definition can hold a minimal set of attributes either to define entity relationships using logical attribute names that are different from the back-end identity store or for the default fetch of attributes.

If an input attribute is not in the identity store schema, the IDS API returns the error thrown by the identity store.

2.2.1.3 Properties of a Logical Entity Definition

You must keep in mind the properties required in each logical entity definition.

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.

2.2.1.4 Properties of a Logical Entity Relationship

You must keep in mind the properties required in each logical entity relationship definition.

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.
`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 for an Identity Directory Service

It is imperative to identify and document the physical characteristics of a configuration item for an Identity Directory, so that it can used as needed.

The following table describes the 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 for an Identity Directory Service

You must explore and identify the functional and operational aspects associated with a configuration item for an Identity Directory Service.

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

The following table describes the 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.

Name	Description
`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 for Identity Directory API

There are some essential design guidelines that one must keep in mind while creating an Identity Directory API.

The following topics describe these recommendations:

- Minimizing Use of defaultFetch Attributes
- Initializing the Identity Directory Once

2.3.1 Minimizing Use of defaultFetch Attributes

You must keep the number of entity attributes to minimal while configuring a new Identity Directory.

The entity attribute is defined by `defaultFetch` value. 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 Initializing the Identity Directory Once

Initialization of Identity Directory has some overhead to initialize the entire ArisId stack. Therefore, you must initialize the Identity Directory once.

The Identity Directory API is used to initialize the Identity Directory Service. It has some overhead. As a result, applications should initialize the Identity Directory once, preferably on application startup, and use only one handle throughout.

2.4 Examples of Using the Identity Directory API

Use the sample codes for performing various operations associated with the Identity Directory API.

The following topics describe operations associated with the Identity Directory API:

- Initializing and Obtaining Identity Directory Handle
- Initializing and Obtaining Identity Directory Handle from JPS Context
- Initializing and Obtaining In-Memory Identity Directory Handle
- Adding a User
- Obtaining a User for Given Principal
- Modifying a User

- Obtaining a User for Given ID Value
- Searching Users Using Complex Search Filter
- Changing User Password
- Resetting User Password
- Authenticating a User
- Deleting a User
- Creating a Group
- Searching Groups
- Obtaining Management Chain
- Obtaining Reportees of a User
- Adding a Member to a Group
- Deleting a Member From a Group
- Obtaining All The Groups For Which User is a Member
- Using Logical NOT Operator in a Search Filter

2.4.1 Initializing and Obtaining Identity Directory Handle

You must first call an initialization function to use the functionality of Identity Directory Service. The Identity Directory handle then obtained is used to perform basic User and Group CRUD operations.

```
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();
    }
}
```

Note:

If you plan to use Tivoli as the authentication provider, then you need to select `OPEN_LDAP` as the authentication provider type. This is because Oracle WebLogic Server does not support Tivoli.

When Identity Governance Framework or Identity Directory Service is initialized to obtain the directory handle for Tivoli, then the generated adapters.os_xml file contains the following parameter:

```
<param name="mapAttribute" value="orclGUID=entryUUID"/>
```

In this scenario, for Tivoli, you need to map `orclGUID` attribute to `ibm-entryUUID` as follows:

```
<param name="mapAttribute" value="orclGUID=ibm-entryUUID"/>
```

You need to update the adapters.os_xml file manually to reflect these changes. In addition, you must restart the Oracle WebLogic Server for any attribute mapping update to work.

2.4.2 Initializing and Obtaining Identity Directory Handle from JPS Context

You can initialize and obtain the Identity Directory handle from JPS context. Use the sample code to perform the task.

```
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 Initializing and Obtaining In-Memory Identity Directory Handle

You can initialize and obtain the in-memory Identity Directory handle. Use the sample code perform this task.

```
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 Adding a User

After obtaining the Identity Directory handle, you can perform CRUD operations on users and groups. Use the sample code to add 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 Obtaining a User for Given Principal

You can retrieve users for a given principal. Use the sample code to perform the task.

```
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 Modifying a User

Once you have created the users, then you can modify the existing attributes of the user or can add attributes by modifying the user. Use the sample code to perform this task.

```
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 Obtaining a User for Given ID Value

You can retrieve the user details based on the identity value of the user. For this you need to create a retrieval query to fetch the details. Use the sample code to perform this task.

```
try {
            ReadOptions readOpts = new ReadOptions();

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

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

2.4.8 Searching Users Using Complex Search Filter

You might have to create complex filters in the user retrieval query, which match the given criteria and return the target search operation results. Use the sample code to perform this task.

```
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 Changing User Password

After you have created a user, you can modify the attributes of a user. Use the sample code to modify the password of a user.

```
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 Resetting User Password

You can reset the password of the created user in Identity Directory. Use the sample code to perform this task.

```
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 Authenticating a User

It is imperative to authenticate a user before granting the access to perform various operations. You can authenticate a user using APIs.

```
        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 Deleting a User

You can delete a user that already exists in the identity store using the Identity Directory API. Use the sample code to perform this task.

```
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 Creating a Group

It is beneficial to create Groups as it is easier to grant or deny privileges to a groups of users instead of applying those privileges to each user individually. You can create user groups in Identity Directory.

```
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 Searching Groups

You can define search filters to search groups matching the desired criteria.

```
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 Obtaining Management Chain

You can obtain the management hierarchy for any given user in Identity Directory. Use the sample code to perform this task.

```
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 Obtaining Reportees of a User

You can obtain the reportees of a user by defining target search filters in Identity Directory.

```
// 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 Adding a Member to a Group

You can logically group an existing user in Identity Directory by adding them to a specific group. Use the sample code to perform this task.

```
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 Deleting a Member From a Group

A user who is a member of a group can be isolated from the given group using the Identity Directory API. Use the sample code to perform this task.

```
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();
}
```

Note:

Identity Governance Framework/Identity Directory Service group membership search evaluates both static and dynamic groups. However, membership updates (addition/deletion) are not supported for dynamic groups. For instance, if you wish to delete a member from a group and the member is a dynamic member of that group, then the delete operation is not supported for the dynamic group.

2.4.19 Obtaining All The Groups For Which User is a Member

For an existing user in Identity Directory, you can obtain all the groups to which the user belongs to using Identity Directory API. Use the sample code to perform this task.

```
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();
        }
```

2.4.20 Using Logical NOT Operator in a Search Filter

Identity Directory supports the use of NOT operator in a search filter. You can easily define a NOT operator in a search filter for obtaining results that match the criteria.

```
try {
     SearchFilter f1 = new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH, "demo");
     SearchFilter f2 = new SearchFilter("orgunit", SearchFilter.Operator.CONTAINS, "myorg");
     f2.negate();
     SearchFilter filter = new SearchFilter(SearchFilter.LogicalOp.AND, f1, f2);

     ResultSet<User> sr = uMgr.searchUsers(filter, searchOpts);
    }
```

3 Using the 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).

The following topics describe the architecture and key functionality of the Identity Governance Framework ArisID API:

- About the ArisID API
- Configuring the ArisID API
- Design Recommendations for the ArisID API
- Generating ArisID Beans
- Examples of Using IDX User and Role Beans
- Understanding OpenLDAP Considerations

3.1 About the 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 12*c* (12.2.2.0) 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.

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.

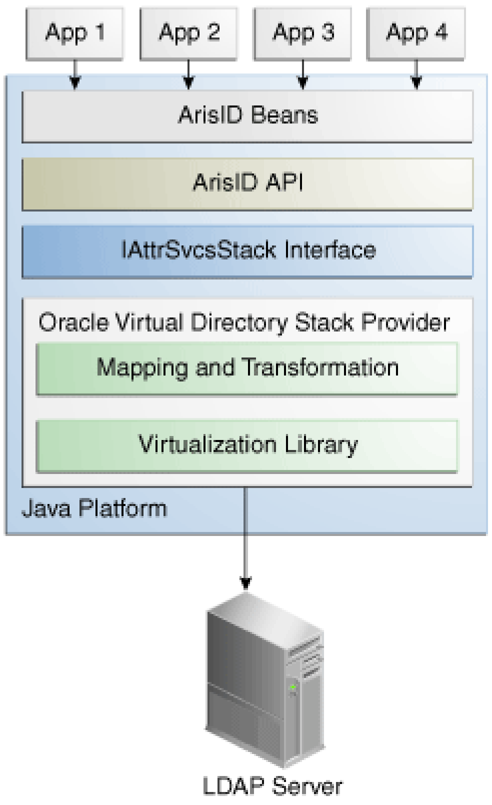
For more information, see *Java API Reference for Identity Governance Framework IDXUserRole*.

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

ArisID API supports several configuration options which allow for easier and flexible deployment in Web Server environments.

The Identity Governance Framework ArisID supports the basic development process, which is 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:

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

3.2.1 Examining CARML Files

CARML is an XML-based declarative format to define the attribute usage requirements of an application. Administrators examine the CARML file to connect to the appropriate back-end resources.

To determine whether the existing ArisID beans meet your application's needs, you need to examine idxuserrole.xml (read-only operations) and userrole.xml (read-only and read/write operations) CARML files. These CARML files are located in DOMAIN_HOME/config/fmwconfig/carml.

3.2.2 Configuring the Identity Repository

An identity repository is a data store where information about users and groups in a company is stored. You can configure the identity repository to be used by the ArisID beans.

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 for Identity Governance Framework.

For more information about Oracle Platform Security Services, see *Oracle® Fusion Middleware Securing Applications with Oracle Platform Security Services*.

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

The mapping file allows you to map CARML attributes to LDAP attributes, objectclasses, and search parameters. You can customize your mapping file for the LDAP-based directory server at your site.

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 for the ArisID API

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 following topics describe the configuration parameters that can be modified:

- About Unique Identifier Configuration
- About Unique Key Attribute Specification
- About Multiple Language Support
- Handling Large Results
- Securing the Application
- Configuring the Timeout Interval
- Specifying Wildcard Characters in Search Filters

3.3.1 About Unique Identifier Configuration

It is imperative to ensure that all users are uniquely identifiable. 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 About Unique Key Attribute Specification

Attribute uniqueness is enforced by defining constraints on attributes.

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 About 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>
```

The `displayname` is the most commonly used multiple language attribute, therefore 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 Handling Large Results

There are situations when you retrieve large result sets. This could cause a performance and resource problem. Use the sample code to manage large result sets.

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 Securing the Application

There are two common security scenarios for executing create, read, update, and delete (CRUD) operations on the target system. You must review these scenarios for a better understanding.

They are as follows:

- Applying Domain Level Credentials
- Applying Application Level Credentials

Proxy authentication is not supported in this release.

3.3.5.1 Applying 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.

Consider the following example where 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 Applying 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

Consider the scenario where 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 Configuring the Timeout Interval

You can configure the timeout intervals for LDAP operations. These intervals specify the amount of time after which the LDAP operation times out.

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 application has a 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.

3.3.7 Specifying Wildcard Characters in Search Filters

Oracle Internet Directory supports wildcard search for DN attributes.

Directories other than Oracle directories might not support wildcard search. Therefore, if you perform a `seachUsers` operation through Identity Governance Framework API with say Microsoft Active Directory as the backend and using a DN in a search filter, then you must provide the `AttributeFilter.OP_EQUALS` operator to successfully execute the operation.

3.4 Generating ArisID Beans

When you have finished editing your CARML file, you can generate the ArisID Beans to use in your application. The ArisID Beans provide Java APIs as a layer on top of the ArisID API that have various action methods associated with them.

To generate the ArisID Beans run the following command:

```
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 Examples of Using IDX User and Role Beans

It is imperative to understanding the use of IDX User and Role Beans. Use the sample applications to understand how to integrate IDX User/Role Beans into your application.

The following topics describe the use of IDX User/Role Beans:

- Searching Users Using IDX User and Role Beans: SearchUsers.jsp
- Searching Users Using IDX User and Role Beans: SearchUsers.html

3.5.1 Searching Users Using IDX User and Role Beans: SearchUsers.jsp

Use IDX User/Role Beans Java version to search users.

```
<%@ 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 Searching Users Using IDX User and Role Beans: SearchUsers.html

Use the IDX User/Role Beans code to search users.

```
<!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 Understanding OpenLDAP Considerations

OpenLDAP specifies some mandatory attributes for implementing the APIs. You must keep in mind these attributes while implementing your applications.

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

Use the topics to understand how to migrate applications from the User and Role API to the Identity Directory API.

- Overview of Migrating to Identity Directory API
- Migrating the Application to Identity Directory API
- Understanding the Comparison Between User and Role API With IDS API
- Moving From a Test to a Production Environment
- Tuning Configuration Parameters for IDS
- Allowing Pass-through Attributes in IDS

4.1 Overview of Migrating to Identity Directory API

The Identity Directory API allows applications to access identity information (users and other entities) in a uniform and portable manner. If you have an application that uses the User and Role API, then you can migrate it to use Identity Directory API.

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 Initializing and Obtaining In-Memory Identity Directory Handle.

4.2 Migrating the Application to Identity Directory API

You need to implement some code changes while migrating an application from the User and Role API to the Identity Directory API.

The following topics describe the code changes needed while migration:

- Initializing API
- Getting UserManager and GroupManager Handle
- Searching Filter
- Performing CRUD Operations

4.2.1 Initializing API

All applications must initialize the API to obtain the Identity Directory handle. The program should perform the initialization only once. Use the sample code to initialize an API.

The process of initializing is similar to using `IdentityStoreService.GetIdmStore()` for getting `oracle.security.idm.IdentityStore` handle. Identity Directory Service 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

All operations on a user instance are handled by a user manager and all operations on a group are handled by a group manager. Use the sample code to perform CRUD operations on users and groups instance respectively.

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 Searching Filter

You can create simple or complex search filter to be used in searching the identity repository. Use the sample code to facilitate a variety of search operations.

You can build a simple or complex search filter 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 Performing CRUD Operations

You can perform Create/Read/Update/Delete (CRUD) operations on User, Group, Org, and generic entities. This requires that the CRUD APIs be implemented for use in the applications.

The following topics describes these CRUD operations:

- APIs to Find a User
- APIs to Search a User
- APIs to Create a User
- APIs to Delete a User
- APIs to Authenticate a User
- APIs to Modify Users and Manage Related Entities

4.2.4.1 APIs to Find 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 APIs to Search a User

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

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

4.2.4.3 APIs to Create a User

You can create a user using an API.

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

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

4.2.4.4 APIs to Delete a User

You can delete a user using an API.

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 APIs to Authenticate a User

It is a common mechanism to authenticate users via an API.

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 APIs to Modify Users and Manage Related Entities

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

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)
  ```

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)
  ```

- 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 Understanding the Comparison Between User and Role API With IDS API

It is essential that you understand the mapping between the User and Role API and Identity Directory API before implementing the change in your application.

The following topics describe the differences:

- Comparison of User-Related APIs With Identity Directory APIs
- Comparison of Role-Related APIs With Identity Directory APIs

4.3.1 Comparison of User-Related APIs With Identity Directory APIs

You must understand the mapping between the endpoints for the User API with those in the Identity Directory API.

The following table maps the User-related API method with its corresponding Identity Directory API method.

Functionality	User/Role API Method	Identity Directory Service Method
User Creation	User UserManager.createUser(String name, char[] password) User UserManager.createUser(String name, char[] password, PropertySet pset)	Principal UserManager.createUser(List<Attribute> attrVals, CreateOptions opts)

Functionality	User/Role API Method	Identity Directory Service Method
Delete User	void UserManager.dropUser(UserProfile user) void UserManager.dropUser(User user);	void UserManager.deleteUser(Principal principal, DeleteOptions opts) void UserManager.deleteUser(String id, DeleteOptions opts)
Authenticate User	User UserManager.authenticateUser(String user_id, char[] passwd) User UserManager.authenticateUser(User user, char[] passwd) User UserManager.authenticateUser(String user_id, String authProperty, char[] passwd)	User UserManager.authenticateUser(String id, char[] password, ReadOptions opts) boolean UserManager.authenticateUser(Principal principal, char[] password)
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)
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)

Functionality	User/Role API Method	Identity Directory Service Method
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.getUserPropertyNames() 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(ModProperty mprop)	void User.setAttributeValue(String attrName, String attrVal, ModifyOptions opts)
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)

Functionality	User/Role API Method	Identity Directory Service Method
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 Comparison of Role-Related APIs With Identity Directory APIs

You must understand the mapping between the endpoints for the User/Role API with those in the Identity Directory API.

The following table maps the Role-related API method with its corresponding Identity Directory API method.

Functionality	User/Role API Method	Identity Directory Service 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(RoleProfile role) void RoleManager.dropRole(Role role)	void GroupManager.deleteGroup(Principal principal, DeleteOptions opts)
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()

Functionality	User/Role API Method	Identity Directory Service Method
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

Functionality	User/Role API Method	Identity Directory Service Method
Add a user as a manager of a Role	void Role.addManager(Principal principal)	Not Supported
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(Search Parameters 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

4.4 Moving From a Test to a Production Environment

Moving from one environment to another, especially from a test environment to production environment, provides you the flexibility to test applications in a test environment and then roll them out in the production environment.

The following topics describe the Identity Directory Service/libOVD properties that you need to modify while moving from a test environment to production environment:

- Overview of Moving Between Environments
- Modifying Identity Directory Service/libOVD Move Plan

4.4.1 Overview of Moving Between Environments

You can move Identity Directory Service/libOVD to a new environment or from a test to a production environment. Moving Identity Directory Service/libOVD installation diminishes the amount of work that would otherwise be required to reapply all the customization and configuration changes made in one environment to another.

You can install, configure, customize, and validate Identity Directory Service/libOVD in a test environment. Once the system is stable and performs as required, you can create the production environment by moving a copy of the server and its configuration from the test environment, instead of redoing all the changes that were incorporated into the test environment.

4.4.2 Modifying Identity Directory Service/libOVD Move Plan

A move plan contains configuration settings of the source environment. You can customize the move plan settings for Oracle Fusion Middleware entities and components.

When you move between environments, you run the extractMovePlan script to create a move plan for the entity that you are moving. The extractMovePlan script extracts configuration information from the archive into a move plan. It also extracts any needed configuration plans. Before you apply the archive to the target, you must edit the move plan to reflect the values of the target environment.

You can modify properties with the scope of READ_WRITE. Do not modify the properties with the scope of READ_ONLY. For a comprehensive description and the procedure to follow for moving between environments, see "Common Procedures for Moving to a Target Environment" in *Administering Oracle Fusion Middleware*.

This section contains the following topics:

- Locating Identity Directory Service/libOVD configGroup Elements
- Properties to Customize for Identity Directory Service/libOVD Move Plan

4.4.2.1 Locating Identity Directory Service/libOVD configGroup Elements

Move plans usually contain multiple configGroup elements. When a property is associated with a particular configGroup element, the tables listing the properties group the properties by configGroup element.

To locate Identity Directory Service/libOVD ConfigGroup, in the generated move plan, you must look for `<type>LIBOVD_ADAPTERS</type>`. This tag provides comprehensive information about the libOVD adapter properties that you might have to update. A property is associated with a particular configGroup element.

Each adapter is represented by a configProperty id tag of the form:

```
"LDAP:<context_name>:<adapter_name>"
```

Consider the following example: `"LDAP:ids:myOID"`

Table 4-1 shows the properties for the move plan for libOVD.

The following example shows a section of the move plan for Identity Directory Service/libOVD, with portion of the `LIBOVD_ADAPTERS` configGroup elements:

```
<configGroup>
                <type>LIBOVD_ADAPTERS</type>
                <configProperty id="LDAP:ids:myOID">
                    <configProperty>
                        <name>Context Name</name>
                        <value>ids</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                            <scope>READ_ONLY</scope>
                        </itemMetadata>
                    </configProperty>
                    <configProperty>
                        <name>Adapter Name</name>
                        <value>myOID</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                            <scope>READ_ONLY</scope>
                        </itemMetadata>
                    </configProperty>
                    <configProperty>
                        <name>LDAP URL</name>
                        <value>ldap://hostname:1389</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                            <scope>READ_WRITE</scope>
                        </itemMetadata>
                    </configProperty>
                    <configProperty>
                        <name>LDAP Host Read Only</name>
                        <value>false</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                            <scope>READ_WRITE</scope>
                        </itemMetadata>
                    </configProperty>
                    <configProperty>
                        <name>LDAP Host Percentage</name>
                        <value>100</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                            <scope>READ_WRITE</scope>
                        </itemMetadata>
                    </configProperty>
                    <configProperty>
                        <name>DN</name>
                        <value>cn=orcladmin</value>
                        <itemMetadata>
                            <dataType>STRING</dataType>
                <scope>READ_WRITE</scope>
            </itemMetadata>
        </configProperty>
        <configProperty>
            <name>Password File</name>
            <value/>
            <itemMetadata>
                <dataType>STRING</dataType>
                <password>true</password>
                <scope>READ_WRITE</scope>
            </itemMetadata>
        </configProperty>
    </configProperty>
</configGroup>
```

4.4.2.2 Properties to Customize for Identity Directory Service/libOVD Move Plan

You can customize the properties of a move plan.

Table 4-1 describes the move plan properties you can customize for Identity Directory Service/libOVD adapter.

Table 4-1 Move Plan Properties for libOVD

Property	Description	Sample Value
Context Name	The libOVD context to use with which the adapter is associated. This is a read-only property.	`ids`
Adapter Name	The name of the adapter. This is a read-only property.	`myOID`
LDAP URL	The LDAP URL value for the adapter in the form of ldap://host:port. This is a read-write property.	`ldap://slc05kym:1389`
DN	The DN of the user to connect to the backend LDAP. This is a read-write property.	`cn=orcladmin`
Password File	The absolute path to the secure file containing the password of the user. This is a read-write property.	`/tmp/p.txt`
LDAP Host Read Only	The flag indicating if the given host is read only. The default value is `false`. This is a read-write property.	`false`
LDAP Host Percentage	It specifies the load percentage value for the given LADAP host. The default value is `100`. This is a read-write property.	`100`

4.5 Tuning Configuration Parameters for IDS

Tuning is the adjustment or modification of parameters to meet specific deployment requirements. The default IDS configuration must be tuned for your deployment scenario.

You must review the requirements and recommendations in this section carefully.

This section contains the following topics:

- Configuration Parameters for IDS
- WLST Commands to Set Tuning Parameters Using File-Based Configuration
- WLST Commands to Set Tuning Parameters Using In-Memory Configuration
- Handling Firewall and Load Balancer Timeout Errors

4.5.1 Configuration Parameters for IDS

You can use configuration parameters to tune performance and to balance memory requirements for a real-time deployment scenario. Tuning these parameters based on your requirements can greatly enhance the scalability characteristics of an application.

Table 4-2 lists the configuration parameters for IDS that require tuning for real deployment scenarios.

Table 4-2 Configuration Parameters for IDS

Parameter	Description
`InitialPoolSize`	The initial number of LDAP connections created when the LDAP connection pool is set up.
`MaxPoolSize`	The maximum number of LDAP connections allowed in the LDAP connection pool. **Note:** If a deployment has numerous concurrent requests coming in, then you must set this value appropriately to prevent running out of connections or waiting for a connection during an operation.
`MaxPoolWait` `MaxPoolChecks`	These parameters determine the waiting time for free LDAP connection when all the LDAP connections in the connection pool are in use. IDS waits for `MAX_POOLWAIT` and `MAX_POOLCHECKS` milliseconds for a free connection to be available first and then tries to expand the connection pool.
`PoolCleanupInterval`	This is the timer interval (in seconds) used by the LDAP connection pool cleanup timer. The LDAP connection pool cleanup timer runs using this timer interval to perform pool cleanup tasks like shrinking the connection pool based on the idle connection if needed.
`MaxPoolConnectionIdleTime`	This specifies the maximum idle time for an LDAP connection. In an LDAP connection remains idle for this amount of time, it will be closed when the next LDAP connection pool cleanup timer runs.
`OperationTimeout`	The amount of time in milliseconds IDS waits for an LDAP request to be acknowledged by the LDAP remote host.
`ConnectTimeout`	This specifies the LDAP connection timeout duration in milli seconds. If a connection cannot be established in this period, then the connection attempt is aborted.

Table 4-2 (Cont.) Configuration Parameters for IDS

Parameter	Description
`HeartbeatInterval`	This is the interval in seconds to check the availability of backend LDAP.
`SocketOptions`	This parameters set `SO_TIMEOUT` (in seconds), `SO_REUSEADDR, TCP_NODELAY, SO_KEEPALIVE` properties for the underlying JNDI sockets in the LDAP connection.
`MaxPoolConnectionReuseTime`	This specifies the maximum time any connection can potentially be reused after which the pool removes and closes a connection. The value is specified in seconds.
`PoolConnectionReclaimTime`	This specifies the time duration in seconds that a borrowed connection can remain unused before it is automatically reclaimed by the pool.
`Protocols`	This specifies the protocol versions supported by IDS.

4.5.2 WLST Commands to Set Tuning Parameters Using File-Based Configuration

The configuration information is stored in an XML file. You must use the WebLogic Scripting Tool (WLST) to modify the tuning parameters using the file-based configuration.

You use the following WLST commands to configure the tuning parameters:

Note:

In all the WLST command examples in this section, `ADAPTER_NAME` refers to the name of the IDS repository. For instance, the libOVD adapter name.

- For `InitialPoolSize`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='InitialPoolSize',
value=10, contextName='ids')
```

- For `MaxPoolSize`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolSize', value=100,
contextName='ids')
```

- For `MaxPoolWait` and `MaxPoolCheck`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolWait',
value=1000, contextName='ids')

modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolChecks',
value=10, contextName='ids')
```

- For `PoolCleanupInterval`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='PoolCleanupInterval',
value=300, contextName='ids')
```

- For `MaxPoolConnectionIdleTime`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME',
attribute='MaxPoolConnectionIdleTime', value=3600, contextName='ids')
```

- For `OperationTimeout`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='OperationTimeout',
value=120000, contextName='ids')
```

- For `ConnectTimeout`

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='ConnectTimeout',
value=10000, contextName='ids')
```

- For `HeartbeatInterval`:

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='HeartBeatInterval',
value=60, contextName='ids')
```

- For `SocketOption`:

```
modifySocketOptions(adapterName='ADAPTER_NAME', reuseAddress=false,
keepAlive=false, tcpNoDelay=true, readTimeout=1800, contextName='ids')
```

- For `MaxPoolConnectionReuseTime`

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME',
attribute='MaxPoolConnectionReuseTime', value=3600, contextName='ids')
```

- For `PoolConnectionReclaimTime`

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME',
attribute='PoolConnectionReclaimTime', value=180, contextName='ids')
```

- For `Protocols`

```
modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='Protocols',
value='TLSv1.2', contextName='ids')
```

Note:

You must run the `activateLibOVDConfigChanges('ids')` WLST command or restart the WebLogic server for configuration changes to take effect.

4.5.3 WLST Commands to Set Tuning Parameters Using In-Memory Configuration

Use the WLST commands to configure the tuning parameters using in-memory configuration.

The configuration information is stored by the IDS consumer and is passed during run-time to IDS by invoking the `IdentityStoreConfig` class. For more information about using the class and its properties, see *Oracle Fusion Middleware Java API Reference for Identity Governance Framework Identity Directory*.

You can modify the following configuration parameters using the Java API class:

Table 4-3 Field Name for Configuration Parameters

Parameter	Field Name to Modify
InitialPoolSize	`IdentityStoreConfig.INITIAL_POOLSIZE`
MaxPoolSize	`IdentityStoreConfig.MAX_POOLSIZE`
MaxPoolWait MaxPoolChecks	`IdentityStoreConfig.MAX_POOLWAIT` `IdentityStoreConfig.MAX_POOLCHECK`
PoolCleanupInterval	`IdentityStoreConfig.POOL_CLEANUP_INTERVAL`
MaxPoolConnectionIdleTime	`IdentityStoreConfig.MAX_POOL_CONNECTION_IDLE_TIME`
OperationTimeout	`IdentityStoreConfig.CONN_TIMEOUT`
ConnectTimeout	`IdentityStoreConfig.CONNECT_TIMEOUT`
HeartbeatInterval	`IdentityStoreConfig.HEARTBEAT_INTERVAL`

SocketOptions	`IdentityStoreConfig.SOCKET_READTIMEOUT`
	`IdentityStoreConfig.SOCKET_REUSEADDRESS`
	`IdentityStoreConfig.SOCKET_KEEPALIVE`
	`IdentityStoreConfig.SOCKET_TCPNODELAY`

MaxPoolConnectionReuseTime	`IdentityStoreConfig.MAX_POOL_CONNECTION_REUSE_TIME`
PoolConnectionReclaimTime	`IdentityStoreConfig.POOL_CONNECTION_RECLAIM_TIME`

4.5.4 Handling Firewall and Load Balancer Timeout Errors

It is imperative to set up timeout on firewalls and load balancers to improve the communication process. It helps to detect issues in a distributed system.

SocketOptions setting helps detect and safely close orphan socket connections caused by remote server failure. TCP waits for the configured duration of time for a response from the remote server before closing the socket. However, when there is a firewall or a Load Balancer between libOVD and the backend LDAP, then you must set the `readTimeout` value in the SocketOptions appropriately to prevent timeout errors. It is recommended that you set this value to a value which is less than the firewall or the Load Balancer timeout.

4.6 Allowing Pass-through Attributes in IDS

In IDS while executing the Search or Update operation, you need to define every attribute that is used by IDS APIs in the entity definition. However, Identity Directory allows you to dynamically add attributes on runtime. These are referred to as the pass-through attributes.

In certain scenarios attributes are specified dynamically. In other words, they could be used in requested attributes or filters without being defined in the entity definition. The pass-through feature implements this usage and does not throw any exception.