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.
## Contents

**Preface** .......................................................................................................................................................... ix

- Audience ............................................................................................................................................................ ix
- Documentation Accessibility ................................................................................................................................. ix
- Related Documents ............................................................................................................................................. ix
- Conventions ......................................................................................................................................................... ix

**What's New in This Guide** .............................................................................................................................. xi

- New Features in Release 12c (12.2.1) .................................................................................................................. xi
- New Features in Release 12c (12.1.3) .................................................................................................................. xi

### Chapter 1

**Introduction to Identity Governance Framework**

1.1 About the Identity Governance Framework ......................................................................................... 1-1

- 1.1.1 Benefits of Identity Governance Framework to Organizations ........................................ 1-1
- 1.1.2 Benefits of Identity Governance Framework to Developers ................................................ 1-2

1.2 About the Identity Governance Framework APIs .................................................................................... 1-2

1.3 System Requirements and Certification for Identity Governance Framework ........................................ 1-3

### Chapter 2

**Using the Identity Directory API**

2.1 An Overview of the Identity Directory API.............................................................................................. 2-1

- 2.1.1 About the Identity Directory API ....................................................................................................... 2-2
- 2.1.2 Identity Directory Service Architecture ............................................................................................... 2-2

2.2 Configuring the Identity Directory API ....................................................................................................... 2-3

- 2.2.1 Logical Entity Configuration for an Identity Directory Service .......................................................... 2-4

  - 2.2.1.1 Properties of a Logical Entity Configuration ................................................................................. 2-4
  - 2.2.1.2 Logical Entity Attributes .............................................................................................................. 2-4

  - 2.2.1.3 Properties of a Logical Entity Definition ....................................................................................... 2-5
  - 2.2.1.4 Properties of a Logical Entity Relationship .................................................................................. 2-5

- 2.2.2 Physical Identity Store Configuration for an Identity Directory Service ............................................ 2-6

- 2.2.3 Operational Configuration for an Identity Directory Service ........................................................... 2-6

2.3 Design Recommendations for the Identity Directory API ........................................................................... 2-6

- 2.3.1 Keep the defaultFetch Attributes Minimal ......................................................................................... 2-6

- 2.3.2 Initialize the Identity Directory Once ................................................................................................... 2-7

2.4 Examples Using the Identity Directory API ................................................................................................ 2-7

- 2.4.1 Initializing and Obtaining Identity Directory Handle ........................................................................... 2-7

- 2.4.2 Initializing and Obtaining Identity Directory Handle from JPS Context ........................................ 2-9
3 Using the ArisID API
3.1 About the ArisID API ................................................................. 3-1
3.2 Configuring the ArisID API .......................................................... 3-2
3.2.1 Examine CARML Files ........................................................... 3-3
3.2.2 Configuring the Identity Repository ....................................... 3-3
3.2.3 Understanding How to Configure the Mapping File .................. 3-3
3.3 Design Recommendations for the ArisID API .............................. 3-4
3.3.1 Choosing a LoginID ............................................................... 3-4
3.3.2 Choosing a UniqueKey ........................................................... 3-4
3.3.3 Specifying Multiple Language Support .................................... 3-5
3.3.4 Handling Large Results ......................................................... 3-5
3.3.5 Securing the Application ....................................................... 3-6
3.3.5.1 Applying Domain Level Credentials ............................... 3-6
3.3.5.2 Applying Application Level Credentials ............................ 3-6
3.3.6 Configuring the Timeout Interval ........................................... 3-7
3.3.7 Specifying Wildcard Characters in Search Filters .................... 3-8
3.4 Generating ArisID Beans ......................................................... 3-8
3.5 Sample Application Using IDX User and Role Beans ................... 3-9
3.5.1 Code Sample 1: SearchUsers.jsp ......................................... 3-9
3.5.2 Code Sample 2: SearchUsers.html ....................................... 3-10
3.6 Understanding OpenLDAP Considerations .................................. 3-11

4 Migrating to Identity Directory API
4.1 An Overview of Migrating to Identity Directory API .................... 4-1
4.2 Migrating the Application to Identity Directory API .................... 4-1
4.2.1 Initializing API ................................................................. 4-2
4.2.2 Getting UserManager and GroupManager Handle .................... 4-2
4.2.3 Searching Filter ............................................................... 4-2
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.4</td>
<td>Performing CRUD Operations</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.4.1</td>
<td>Finding a User</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.4.2</td>
<td>Searching a User</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.4.3</td>
<td>Creating a User</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.4.4</td>
<td>Deleting a User</td>
<td>4-3</td>
</tr>
<tr>
<td>4.2.4.5</td>
<td>Authenticating a User</td>
<td>4-4</td>
</tr>
<tr>
<td>4.2.4.6</td>
<td>Modifying Users and Managing Related Entities</td>
<td>4-4</td>
</tr>
<tr>
<td>4.3</td>
<td>Understanding the Comparison Between User and Role API With IDS API</td>
<td>4-5</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Comparing User-Related APIs With Identity Directory APIs</td>
<td>4-5</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Comparing Role-Related APIs With Identity Directory APIs</td>
<td>4-7</td>
</tr>
<tr>
<td>4.4</td>
<td>Moving From a Test to a Production Environment</td>
<td>4-9</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Overview of Moving Between Environments</td>
<td>4-10</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Modifying Identity Directory Service/libOVD Move Plan</td>
<td>4-10</td>
</tr>
<tr>
<td>4.4.2.1</td>
<td>Locating Identity Directory Service/libOVD configGroup Elements</td>
<td>4-10</td>
</tr>
<tr>
<td>4.4.2.2</td>
<td>Understanding Identity Directory Service/libOVD Move Plan Properties</td>
<td>4-11</td>
</tr>
<tr>
<td>4.5</td>
<td>Tuning Configuration Parameters for IDS</td>
<td>4-12</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Understanding Configuration Parameters</td>
<td>4-12</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Setting Tuning Parameters Using File-Based Configuration</td>
<td>4-13</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Setting Tuning Parameters Using In-Memory Configuration</td>
<td>4-14</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Handling Firewall and Load Balancer Timeout Errors</td>
<td>4-15</td>
</tr>
</tbody>
</table>
List of Figures

2–1 Identity Directory API Architecture........................................................................................ 2-3
2–2 Identity Directory API Components ........................................................................................ 2-3
3–1 IGF ArisID API Architecture.................................................................................................... 3-2
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 FrameworkIDXUserRole
- Java API Reference for Identity Governance Framework UserRole
- Securing Applications with Oracle Platform Security Services
- 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:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
What's New in This Guide

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

New Features in Release 12c (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 Section 4.5.1, "Understanding Configuration Parameters."

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, using the PoolConnectionReclaimTime configuration parameter. 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 Section 4.5.1, "Understanding Configuration Parameters."

New Features in Release 12c (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 Section 2.4.20, "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 identity store schema, the IDS API returns the error thrown by the identity store.

See Section 2.2.1, "Logical Entity Configuration for an Identity Directory Service."
Introduction to Identity Governance Framework

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

This chapter contains the following topics:

- About the Identity Governance Framework
- About the Identity Governance Framework APIs
- System Requirements and Certification for Identity Governance Framework

1.1 About the Identity Governance Framework

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

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

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

This section contains the following topics:

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

1.1.1 Benefits of Identity Governance Framework to Organizations

Organizations need to maintain control and integrity of sensitive personal information about their customers, employees, and partners. Data related to social security numbers, credit card numbers, medical history and more are increasingly under scrutiny by regulations seeking to prevent abuse or theft of such information. Privacy
conscious organizations frequently have reacted to these requirements by enforcing overly strict controls and processes that hinder business operations and impact productivity, flexibility, and efficiency. At the opposite end of the spectrum, some organizations do not take the care needed to safeguard this information, potentially putting identity-related data at risk without sufficient oversight and control. The Identity Governance Framework enables a standards-based mechanism for enterprises to establish "contracts" between their applications so that identity related information can be shared securely and with confidence that this data will not be abused, compromised, or misplaced. Using this framework, organizations have complete visibility into how identity information is stored, used, and propagated throughout their business. This enables organizations to automate controls to streamline business processes without fear of compromising the confidentiality of sensitive identity related information.

1.1.2 Benefits 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 standards-based way to easily write applications that use this data so that governing policies can be used to control it. This will result in faster development of privacy aware applications.

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

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

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

1.2 About the Identity Governance Framework APIs

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

- **Identity Directory API**

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

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

1.3 System Requirements and Certification for Identity Governance Framework

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

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches. 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.
This chapter describes the architecture and key functionality of the Identity Governance Framework Identity Directory API (Identity Directory API) and Identity Directory Service. The Identity Directory API supports accessing and managing users, groups, organizations, and can be extended to support new entity types with relationships defined between these entities.

This chapter contains the following topics:

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

See Also: Securing Applications with Oracle Platform Security Services

2.1 An 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 fully configurable by clients supporting heterogeneous identity stores having standard and specific schemas, and is robust with both high-availability and failover support.

The API uses the Identity Governance Framework and provides all the benefits of Identity Governance 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 Chapter 1, "Introduction to Identity Governance Framework".

The API supports the following actions:

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

### 2.1.1 About the 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 *Securing Applications with Oracle Platform Security Services*.

### 2.1.2 Identity Directory Service Architecture

*Figure 2–1* shows the logical architecture of the Identity Directory API.
2.2 Configuring the Identity Directory API

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

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

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

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

DOMAIN_HOME/config/fmwconfig

This section contains the following topics:

- Logical Entity Configuration 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

This section describes the logical entity configuration information for an Identity Directory Service.

It contains the following topics:

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

#### 2.2.1.1 Properties of a Logical Entity Configuration

The table in this section describes the properties of a logical entity configuration.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name that uniquely identifies the Identity Directory Service.</td>
</tr>
<tr>
<td>ovd.context</td>
<td>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.</td>
</tr>
<tr>
<td>app.name</td>
<td>Optional property to specify the specific application for which the Identity Directory Service is being configured.</td>
</tr>
</tbody>
</table>

#### 2.2.1.2 Logical Entity Attributes

The table in this section describes the logical entity attributes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Logical attribute name.</td>
</tr>
<tr>
<td>dataType</td>
<td>Valid data type values are as follows: string, boolean, integer, double, datetime, binary, x500name, and rfc822name.</td>
</tr>
<tr>
<td>description</td>
<td>Detailed description of the logical attribute.</td>
</tr>
<tr>
<td>readOnly</td>
<td>Default is false. Use true if the attribute is read-only.</td>
</tr>
<tr>
<td>pwdAttr</td>
<td>Default is false. Use true if the attribute is a password attribute.</td>
</tr>
</tbody>
</table>
Note: Beginning with the 12c (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 Section 2.2.1.2, "Logical Entity Attributes") and in attribute references under the entity definition (described in table in Section 2.2.1.3, "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

The table in this section describes the properties required in each logical entity definition.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the entity.</td>
</tr>
<tr>
<td>type</td>
<td>Valid entity values are as follows: user, group, org and other.</td>
</tr>
<tr>
<td>idAttr</td>
<td>Logical attribute that uniquely identifies the entity.</td>
</tr>
<tr>
<td>create</td>
<td>Use true if creating this entity is allowed. Use false otherwise.</td>
</tr>
<tr>
<td>modify</td>
<td>Use true if modifying this entity is allowed. Use false otherwise.</td>
</tr>
<tr>
<td>delete</td>
<td>Use true if deleting this entity is allowed. Use false otherwise.</td>
</tr>
<tr>
<td>search</td>
<td>Use true if search of this entity to be allowed. Use false otherwise.</td>
</tr>
</tbody>
</table>

**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 less-than. Value none means no filter support.

### 2.2.1.4 Properties of a Logical Entity Relationship

The table in this section describes the properties required in each logical entity relationship definition.
### 2.2.2 Physical Identity Store Configuration for an Identity Directory Service

The table in this section describes the physical identity store configuration properties.

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

### 2.2.3 Operational Configuration 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.

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

### 2.3 Design Recommendations for the Identity Directory API

This section describes the recommendations that one should keep in mind while designing a new Identity Directory API. It contains the following topics:

- Keep the defaultFetch Attributes Minimal
- Initialize the Identity Directory Once

#### 2.3.1 Keep the defaultFetch Attributes Minimal

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

### 2.3.2 Initialize the Identity Directory Once

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

### 2.4 Examples Using the Identity Directory API

This section provides the following code samples:

- 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

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

```java
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:

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

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

```xml
<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

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

```java
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

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

```java
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.CREATE
_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.CREATE
_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();
2.4.4 Adding a User

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

```java
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

The following code sample obtains a user for a given principal:

```java
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

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

```java
try {
    ModifyOptions modifyOpts = new ModifyOptions();
    // Code to modify user attributes goes here
}
```
Examples Using the Identity Directory API

2.4.7 Obtaining a User for Given ID Value

The following code sample obtains a user matching the given identity value:

```java
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

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

```java
try {
    // Complex search filter with nested AND and OR conditions
    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);
```
2.4.9 Changing User Password

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

```java
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

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

```java
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

The following code sample shows how to authenticate a user:

```java
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

The following code sample shows how to delete a user:

```java
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

The following code sample shows how to create a group:

```java
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

The following code sample shows how to search groups:

```java
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

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

```java
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

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

```java
// 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

The following code sample adds a member to a group:

```java
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

The following code sample deletes a member from a group:

```java
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

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

```java
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

The following code sample shows how to use the `negate()` method in a search filter:

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

This chapter contains the following topics:

- **About the ArisID API**
- **Configuring the ArisID API**
- **Design Recommendations for the ArisID API**
- **Generating ArisID Beans**
- **Sample Application 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 12c (12.2.1) release of the ArisID API is a subset of the configuration proposed at:


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:
3.2 Configuring the ArisID API

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

This section contains the following topics:
3.2.1 Examine CARML Files

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

3.2.2 Configuring the Identity Repository

The identity repository to be used by the ArisID beans must be available. You can use the Oracle WebLogic Server embedded LDAP-based directory server or any LDAP directory supported by 11g 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 Section 1.3, "System Requirements and Certification for Identity Governance Framework".

For more information about Oracle Platform Security Services, see 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 adapters.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 Understanding How to Configure the Mapping File

When a CARML file is created a corresponding mapping file is created in the same location. The default mapping file has attribute details specific to Oracle WebLogic Server embedded directory server, which is the Oracle Platform Security Services default identity store. If you are using a default CARML file and the Oracle Platform Security Services identity store, you do not need to configure mapping. The configuration parameters in Oracle Platform Security Services override the parameters in the mapping file.
3.3 Design Recommendations 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 configuration parameters that can be modified are discussed in this section.

This section contains the following topics:

- Choosing a LoginID
- Choosing a UniqueKey
- Specifying Multiple Language Support
- Handling Large Results
- Securing the Application
- Configuring the Timeout Interval
- Specifying Wildcard Characters in Search Filters

3.3.1 Choosing a LoginID

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

```java
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 Choosing a UniqueKey

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

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

```java
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:
3.3.3 Specifying 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>...</multiLanguageAttributes>
```

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

**Restrictions**

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

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

3.3.4 Handling Large Results

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

The following example shows a typical usage pattern:

```java
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

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

- 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.

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

While initializing the ArisID API do not provide any credentials.

```java
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&lt;PropertyFilterValue&gt; attrFilters = new ArrayList&lt;PropertyFilterValue&gt;();
attrFilters.add(new PropertyFilterValue("User.FIRSTNAME", "app1", AttributeFilter.OP_CONTAINS));
attrFilters.add(new PropertyFilterValue("User.LASTNAME", "user1", AttributeFilter.OP_BGNSWITH));
Map&lt;String, Object&gt; 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.

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

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

```java
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

The default connect/read timeout 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)`

3.3.7 Specifying Wildcard Characters in Search Filters

Oracle Internet Directory supports wildcard search for DN attributes. However, directories other than Oracle directories might not support wildcard search. Therefore, if you perform a searchUsers 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

This section describes the steps that you must perform to generate ArisID beans.

Run the following command as follows:

```java
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:

```xml
<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/arisdprovider/conf, and
   b. Copy the CARML file under DOMAIN_HOME/config/fmwconfig/carml

3.5 Sample Application Using IDX User and Role Beans

This section describes the sample applications that use IDX User/Role Beans. They are as follows:

- Code Sample 1: SearchUsers.jsp
- Code Sample 2: SearchUsers.html

3.5.1 Code Sample 1: SearchUsers.jsp

The following code sample allows you to search users using IDX User/Role Beans:

```java
<%@ 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();
   }  
%

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);

<a href="SearchUsers.html">Home</a>

List of Users with FirstName starting with "<%=firstname%>", LastName starting with "<%=lastname%>" and TelephoneNumber containing "<%=telephone%>">

Iterator<User> sIter = subjs.iterator();
while (sIter.hasNext()) {
    User subj = sIter.next();
    Map<String, IAttributeValue> vals = subj.getAllAttributes();
    Iterator<IAttributeValue> iter = vals.values().iterator();
    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) {
            %>
            <table border="0">
                <tr>
                    <th>Item</th>
                    <th>Value</th>
                </tr>
                <tr>
                    <td><%=name%></td>
                    <td><%=value%></td>
                </tr>
            </table>
        %>
    }
}</body>
</html>

3.5.2 Code Sample 2: SearchUsers.html

The following code sample allows you to search users using IDX User/Role Beans:

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3.6 Understanding OpenLDAP Considerations

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

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

If the `Role.MEMBER` is not included in the input `attrVals` list, role creation will fail.
This chapter describes how to migrate applications from the User and Role API to the Identity Directory API.

This chapter contains the following topics:

- An 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

4.1 An Overview of Migrating to Identity Directory API

If you have an application that uses the User and Role API described in Securing Applications with Oracle Platform Security Services and Java API Reference for Oracle Platform Security Services User and Role, you can modify it to use Identity Directory API instead.

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

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

4.2 Migrating the Application to Identity Directory API

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

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

The process of initializing is similar to using `IdentityStoreService.GetIdmStore()` for getting `oracle.security.idm.IdentityStore` handle. Identity Directory API uses `IdentityStoreService.getIdentityStore()` to get `IdentityDirectory` handle. For example:

```java
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);
IdentityDirectory ids = idstore.getIdentityStore();
```

4.2.2 Getting UserManager and GroupManager Handle

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

```java
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

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

```java
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

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

- Finding a User
- Searching a User
- Creating a User
- Deleting a User
- Authenticating a User
- Modifying Users and Managing Related Entities

4.2.4.1 Finding a User

The following APIs are used for finding a user:

- Get user for given principal identifier. For example:
  ```java
  User getUser(Principal principal, ReadOptions opts)
  ```

- Search for user matching given id attribute value that uniquely identifies the user. For example:
  ```java
  User searchUser(String id, ReadOptions opts)
  ```

- Finds user matching given attribute name and value. For example:
  ```java
  User searchUser(String attrName, String attrVal, ReadOptions opts)
  ```

- Search for user matching given GUID value that uniquely identifies the user. For example:
  ```java
  searchUserByGuid(String guid, ReadOptions opts)
  ```

4.2.4.2 Searching a User

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

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

4.2.4.3 Creating a User

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

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

4.2.4.4 Deleting a User

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

- Delete the user given the principal identifier.
  ```java
  void deleteUser(Principal principal, DeleteOptions opts)
  ```

- Delete the user given the id attribute value.
void deleteUser(String id, DeleteOptions opts)

### 4.2.4.5 Authenticating a User
The following are examples of the API for user authentication.

- **Authenticate the user matching the given id attribute value.**
  
  User authenticateUser(String id, char[] password, ReadOptions opts)

- **Authenticate the user for given principal identifier.**
  
  boolean authenticateUser(Principal principal, char[] password)

### 4.2.4.6 Modifying Users and Managing Related Entities
The APIs for modifying user attributes and for getting the related entities are in User object instead of UserManager.

#### 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)
4.3 Understanding the Comparison Between User and Role API With IDS API

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

- Comparing User-Related APIs With Identity Directory APIs
- Comparing Role-Related APIs With Identity Directory APIs

4.3.1 Comparing User-Related APIs With Identity Directory APIs

The table in this section provides a comparison between the User-related API method and the corresponding Identity Directory API method.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>User/Role API Method</th>
<th>Identity Directory API Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Creation</td>
<td>User</td>
<td>Principal</td>
</tr>
<tr>
<td></td>
<td>UserManager.createUser(String name, char[] password)</td>
<td>UserManager.createUser(List&lt;Attribute&gt; attrVals, CreateOptions opts)</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Principal</td>
</tr>
<tr>
<td></td>
<td>UserManager.createUser(String name, char[] password, PropertySet pset)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Principal</td>
</tr>
<tr>
<td></td>
<td>UserManager.createUser(List&lt;Attribute&gt; attrVals, CreateOptions opts)</td>
<td></td>
</tr>
<tr>
<td>Delete User</td>
<td>void</td>
<td>void UserManager.deleteUser(Principal principal, DeleteOptions opts)</td>
</tr>
<tr>
<td></td>
<td>void UserManager.dropUser(UserProfile user)</td>
<td>void UserManager.deleteUser(String id, DeleteOptions opts)</td>
</tr>
<tr>
<td></td>
<td>void UserManager.dropUser(User user)</td>
<td></td>
</tr>
<tr>
<td>Authenticate User</td>
<td>User</td>
<td>User</td>
</tr>
<tr>
<td></td>
<td>UserManager.authenticateUser(String user_id, char[] passwd)</td>
<td>UserManager.authenticateUser(String id, char[] password, ReadOptions opts)</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>UserManager.authenticateUser(User user, char[] passwd)</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td>UserManager.authenticateUser(String user_id, String authProperty, char[] passwd)</td>
<td>boolean</td>
</tr>
<tr>
<td>Check if create User is supported</td>
<td>boolean UserManager.isCreateUserSupported()</td>
<td>boolean UserManager.getCapabilities().isCreateCapable()</td>
</tr>
</tbody>
</table>

Add this user as a member to given group.

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

Delete this user as a member to given group.

```java
void deleteMemberOf(Group group, ModifyOptions opts)
```
<table>
<thead>
<tr>
<th>Functionality</th>
<th>User/Role API Method</th>
<th>Identity Directory API Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if modify User is supported</td>
<td>boolean UserManager.isModifyUserSupported()</td>
<td>boolean UserManager.getCapabilities().isUpdateCapable()</td>
</tr>
<tr>
<td>Check if drop User is supported</td>
<td>boolean UserManager.isDropUserSupported()</td>
<td>boolean UserManager.getCapabilities().isDeleteCapable()</td>
</tr>
<tr>
<td>Search Users by given search criteria</td>
<td>SearchResponse IdentityStore.searchUsers(SearchParameters params)</td>
<td>ResultSet&lt;User&gt; UserManager.searchUsers(SearchFilter filter, SearchOptions opts)</td>
</tr>
<tr>
<td>Search an User by name/uniquename/guid</td>
<td>User IdentityStore.searchUser(String name)</td>
<td>UserManager.searchUser(String id, ReadOptions opts)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User UserManager.searchUser(String attrName, String attrVal, ReadOptions opts)</td>
</tr>
<tr>
<td>Check if User exists in the repository for a given User object</td>
<td>boolean IdentityStore.exists(User user)</td>
<td>User.getPrincipal() if the following method returns null user doesn’t exist; otherwise exists User.getUser(Principal principal, ReadOptions opts)</td>
</tr>
<tr>
<td>Simple search filter (search based on a single attribute name, type and value)</td>
<td>SimpleSearchFilter</td>
<td>SearchFilter(String propertyName, Operator op, String propertyVal)</td>
</tr>
<tr>
<td>Complex Search Filter (search based on more than one attribute with filter conditions and nested filters)</td>
<td>ComplexSearchFilter</td>
<td>SearchFilter(LogicalOp op, SearchFilter... searchFilters)</td>
</tr>
<tr>
<td>Getting a property value for a given property name</td>
<td>String User.getPropertyVal(String propName)</td>
<td>String User.getAttributeValue(String attrName)</td>
</tr>
<tr>
<td></td>
<td>(User Role API fetches the attribute values from cache. If it misses cache, it fetches from repository)</td>
<td>Limitation: Returns attribute values from User object that has been already fetched from the repository.</td>
</tr>
<tr>
<td>Getting the User property for a given property name</td>
<td>Property User.getProperty(String propName)</td>
<td>Attribute User.getAttribute(String attrName)</td>
</tr>
<tr>
<td>Getting the user properties for a given set of property names</td>
<td>Map User.getProperties()</td>
<td>Map&lt;String, Attribute&gt; User.getAllAttributes()</td>
</tr>
<tr>
<td>Get all user properties from the repository for a user</td>
<td>PropertySet User.getAllUserProperties()</td>
<td>Map&lt;String, Attribute&gt; User.getAllAttributes()</td>
</tr>
<tr>
<td>Get all user property names from the schema</td>
<td>List IdentityStore.getUserPropertyNames()</td>
<td>List&lt;String&gt; UserManager.getEntityAttributes()</td>
</tr>
<tr>
<td></td>
<td>Returns the names of all the properties in the schema</td>
<td></td>
</tr>
</tbody>
</table>
4.3.2 Comparing Role-Related APIs With Identity Directory APIs

The table in this section provides a comparison between the Role-related API method and the corresponding Identity Directory API method.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>User/Role API Method</th>
<th>Identity Directory API Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changing the attribute value</td>
<td>void User.setProperty(ModProp</td>
<td>void User.setAttributeValue(String attrName, String attrVal, ModifyOptions opts)</td>
</tr>
<tr>
<td>in the repository of an user</td>
<td>erty mprop)</td>
<td></td>
</tr>
<tr>
<td>Changing the set of attribute</td>
<td>void User.setProperties(ModPro</td>
<td>void User.modify(List&lt;ModAttribute&gt; attrs, ModifyOptions opts)</td>
</tr>
<tr>
<td>values in the repository for</td>
<td>perty[]) modPropObjs)</td>
<td></td>
</tr>
<tr>
<td>an user</td>
<td>void User.setProperties(LdapCo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>text ctx, ModProperty[]) mod</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PropObjs)</td>
<td></td>
</tr>
<tr>
<td>Get all the reportees of an</td>
<td>SearchResponse User.getReportees(boolean direct)</td>
<td>ResultSet&lt;User&gt; User.getReportees(int nLevels, SearchFilter targetFilter, SearchOptions opts)</td>
</tr>
<tr>
<td>User either direct or indirect</td>
<td>User.getReportees(boolean direct)</td>
<td></td>
</tr>
<tr>
<td>Get Management chain of an</td>
<td>List User.getManagementChain</td>
<td>ResultSet&lt;User&gt; User.getManagementChain(int nLevels, SearchOptions opts)</td>
</tr>
<tr>
<td>User</td>
<td>(int max, String upToManagerName, String upToTitle)</td>
<td>List&lt;User&gt; User.getManagementChain(int nLevels, String manager, String title, SearchOptions opts)</td>
</tr>
<tr>
<td>Get/Set of Binary Attributes</td>
<td>Available Property in User/Role API supports binary attributes</td>
<td>Returns base64 encoded value</td>
</tr>
<tr>
<td></td>
<td>byte[] user.getJPEGPhoto()</td>
<td>While setting the value either base64 encoded value or byte[] can be used for creating ModAttribute.</td>
</tr>
<tr>
<td></td>
<td>void user.setJPEGPhoto(String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>imgpath)</td>
<td></td>
</tr>
<tr>
<td>Selecting the Realm</td>
<td>Available env.put(OIDIdentityStoreFactory.RT_SUBSCRIBER_NAME, &quot;&lt;realm dn&gt;&quot;); IdentityStoreFactory.getIdentityStoreInstance(env);</td>
<td>This is part of IDS Operational configuration. At API level searchbase and createbase can be specified as well.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functionality</th>
<th>User/Role API Method</th>
<th>Identity Directory API Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Role</td>
<td>Role RoleManager.createRole(String name, int scope)</td>
<td>Principal GroupManager.createGroup(List&lt;Attribute&gt; attrVals, CreateOptions opts)</td>
</tr>
<tr>
<td></td>
<td>Role RoleManager.createRole(String name)</td>
<td></td>
</tr>
<tr>
<td>Deleting a Role</td>
<td>void RoleManager.dropRole(Role Profile role)</td>
<td>void GroupManager.deleteGroup(Principal principal, DeleteOptions opts)</td>
</tr>
<tr>
<td></td>
<td>void RoleManager.dropRole(Role role)</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>User/Role API Method</td>
<td>Identity Directory API Method</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----------------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Check if create role is supported</td>
<td>boolean RoleManager.isCreateRoleSupported()</td>
<td>boolean GroupManager.getCapabilities().isCreateCapable()</td>
</tr>
<tr>
<td>Check if modify role is supported</td>
<td>boolean RoleManager.isModifyRoleSupported()</td>
<td>boolean GroupManager.getCapabilities().isUpdateCapable()</td>
</tr>
<tr>
<td>Check if delete role is supported</td>
<td>boolean RoleManager.isDropRoleSupported()</td>
<td>boolean GroupManager.getCapabilities().isDeleteCapable()</td>
</tr>
<tr>
<td>Is the Group owned by a User</td>
<td>boolean RoleManager.isGranted(Role parent, Principal principal)</td>
<td>boolean Group.isMember(User user, boolean direct, ReadOptions opts)</td>
</tr>
<tr>
<td>Is the Group owned by a User</td>
<td>boolean RoleManager.isOwnedBy(Role parent, Principal principal)</td>
<td>boolean User.isOwnerOf(Group group, boolean direct, ReadOptions opts)</td>
</tr>
<tr>
<td>Is the group managed by a User</td>
<td>boolean RoleManager.isManagedBy(Role parent, Principal principal)</td>
<td>Not supported</td>
</tr>
<tr>
<td>Get all the members of a Role either direct / indirect</td>
<td>SearchResponse Role.getGrantees(SearchFilter filter, boolean direct)</td>
<td>ResultSet&lt;User&gt; Group.getMembers(int nLevels, SearchFilter targetFilter, SearchOptions opts)</td>
</tr>
<tr>
<td>Add an user as a member to a role</td>
<td>void RoleManager.grantRole(Role parent, Principal principal)</td>
<td>void Group.addMember(User user, ModifyOptions opts)</td>
</tr>
<tr>
<td>Remove a user from being member of a role</td>
<td>void RoleManager.revokeRole(Role parent, Principal principal)</td>
<td>void Group.deleteMember(User user, ModifyOptions opts)</td>
</tr>
<tr>
<td>Get all the owners of a specific Role either direct / indirect</td>
<td>SearchResponse Role.getOwners(SearchFilter filter, boolean direct)</td>
<td>ResultSet&lt;User&gt; Group.getOwners(int nLevels, SearchFilter targetFilter, SearchOptions opts)</td>
</tr>
<tr>
<td>Add a user as a owner of a role</td>
<td>void Role.addOwner(Principal principal)</td>
<td>void Group.addOwner(User user, ModifyOptions opts)</td>
</tr>
<tr>
<td>Remove a user from being a owner of a Role</td>
<td>void Role.removeOwner(Principal principal)</td>
<td>void Group.deleteOwner(User user, ModifyOptions opts)</td>
</tr>
<tr>
<td>Get all the managers of a Role either direct / indirect</td>
<td>SearchResponse Role.getManagers(SearchFilter filter, boolean direct)</td>
<td>Not Supported</td>
</tr>
<tr>
<td>Add a user as a manager of a Role</td>
<td>void Role.addManager(Principal principal)</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>
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.

This section describes the Identity Directory Service/libOVD properties that you need to modify while moving from a test environment to production environment, and contains the following topics:

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

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. 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 "Moving from a Test to a Production Environment" section in the Oracle Fusion Middleware Administrator’s Guide.

This section contains the following topics:

- Locating Identity Directory Service/libOVD configGroup Elements
- Understanding Identity Directory Service/libOVD Move Plan Properties

4.4.2.1 Locating Identity Directory Service/libOVD configGroup Elements

Move plans usually contain multiple configGroup elements. 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>
      </itemMetadata>
    </configProperty>
  </configProperty>
</configGroup>
```
4.4.2.2 Understanding Identity Directory Service/libOVD Move Plan Properties

The Table 4–1 in this section describes the move plan properties you can customize for Identity Directory Service/libOVD adapter.
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:

- Understanding Configuration Parameters
- Setting Tuning Parameters Using File-Based Configuration
- Setting Tuning Parameters Using In-Memory Configuration
- Handling Firewall and Load Balancer Timeout Errors

4.5.1 Understanding Configuration Parameters

Table 4–2 lists the configuration parameters that require tuning in a real deployment scenario for IDS.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialPoolSize</td>
<td>The initial number of LDAP connections created when the LDAP connection pool is set up.</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>The maximum number of LDAP connections allowed in the LDAP connection pool.</td>
</tr>
</tbody>
</table>

**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.
4.5.2 Setting Tuning Parameters Using File-Based Configuration

The configuration information is stored in XML file. In such a scenario, you can use the WebLogic Scripting Tool (WLST) to modify the configuration parameters.

This section describes how to configure the tuning parameters using the WLST command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxPoolWait</td>
<td>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.</td>
</tr>
<tr>
<td>MaxPoolChecks</td>
<td></td>
</tr>
<tr>
<td>PoolCleanupInterval</td>
<td>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.</td>
</tr>
<tr>
<td>MaxPoolConnectionIdleTime</td>
<td>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.</td>
</tr>
<tr>
<td>OperationTimeout</td>
<td>The amount of time in milliseconds IDS waits for an LDAP request to be acknowledged by the LDAP remote host.</td>
</tr>
<tr>
<td>ConnectTimeout</td>
<td>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.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>This is the interval in seconds to check the availability of backend LDAP.</td>
</tr>
<tr>
<td>SocketOptions</td>
<td>This parameters set SO_TIMEOUT (in seconds), SO_REUSEADDR, TCP_NODELAY, SO_KEEPALIVE properties for the underlying JNDI sockets in the LDAP connection.</td>
</tr>
<tr>
<td>MaxPoolConnectionReuseTime</td>
<td>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.</td>
</tr>
<tr>
<td>PoolConnectionReclaimTime</td>
<td>This specifies the time duration in seconds that a borrowed connection can remain unused before it is automatically reclaimed by the pool.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxPoolWait</td>
<td>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.</td>
</tr>
<tr>
<td>MaxPoolChecks</td>
<td></td>
</tr>
<tr>
<td>PoolCleanupInterval</td>
<td>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.</td>
</tr>
<tr>
<td>MaxPoolConnectionIdleTime</td>
<td>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.</td>
</tr>
<tr>
<td>OperationTimeout</td>
<td>The amount of time in milliseconds IDS waits for an LDAP request to be acknowledged by the LDAP remote host.</td>
</tr>
<tr>
<td>ConnectTimeout</td>
<td>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.</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>This is the interval in seconds to check the availability of backend LDAP.</td>
</tr>
<tr>
<td>SocketOptions</td>
<td>This parameters set SO_TIMEOUT (in seconds), SO_REUSEADDR, TCP_NODELAY, SO_KEEPALIVE properties for the underlying JNDI sockets in the LDAP connection.</td>
</tr>
<tr>
<td>MaxPoolConnectionReuseTime</td>
<td>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.</td>
</tr>
<tr>
<td>PoolConnectionReclaimTime</td>
<td>This specifies the time duration in seconds that a borrowed connection can remain unused before it is automatically reclaimed by the pool.</td>
</tr>
</tbody>
</table>

### 4.5.2 Setting Tuning Parameters Using File-Based Configuration

The configuration information is stored in XML file. In such a scenario, you can use the WebLogic Scripting Tool (WLST) to modify the configuration parameters.

This section describes how to configure the tuning parameters using the WLST command.

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

---

Migrating to Identity Directory API 4-13
- For **MaxPoolSize**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolSize', value=100, contextName='ids')
  ```

- For **MaxPoolWait** and **MaxPoolCheck**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolWait', value=1000, contextName='ids')
  
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolChecks', value=10, contextName='ids')
  ```

- For **PoolCleanupInterval**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='PoolCleanupInterval', value=300, contextName='ids')
  ```

- For **MaxPoolConnectionIdleTime**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolConnectionIdleTime', value=3600, contextName='ids')
  ```

- For **OperationTimeout**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='OperationTimeout', value=120000, contextName='ids')
  ```

- For **ConnectTimeout**
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='ConnectTimeout', value=10000, contextName='ids')
  ```

- For **HeartbeatInterval**:
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='HeartBeatInterval', value=60, contextName='ids')
  ```

- For **SocketOption**:
  
  ```java
  modifySocketOptions(adapterName='ADAPTER_NAME', reuseAddress=false, keepAlive=false, tcpNoDelay=true, readTimeout=1800, contextName='ids')
  ```

- For **MaxPoolConnectionReuseTime**
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='MaxPoolConnectionReuseTime', value=3600, contextName='ids')
  ```

- For **PoolConnectionReclaimTime**
  
  ```java
  modifyLDAPAdapter(adapterName='ADAPTER_NAME', attribute='PoolConnectionReclaimTime', value=180, 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 Setting 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>
<thead>
<tr>
<th>Parameter</th>
<th>Field Name to Modify</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialPoolSize</td>
<td>IdentityStoreConfig.INITIAL_POOLSIZE</td>
</tr>
<tr>
<td>MaxPoolSize</td>
<td>IdentityStoreConfig.MAX_POOLSIZE</td>
</tr>
<tr>
<td>MaxPoolWait</td>
<td>IdentityStoreConfig.MAX_POOLWAIT</td>
</tr>
<tr>
<td>MaxPoolChecks</td>
<td>IdentityStoreConfig.MAX_POOLCHECK</td>
</tr>
<tr>
<td>PoolCleanupInterval</td>
<td>IdentityStoreConfig.POLL_CLEANUP_INTERVAL</td>
</tr>
<tr>
<td>MaxPoolConnectionIdleTime</td>
<td>IdentityStoreConfig.MAX_POOL_CONNECTION_IDLE_TIME</td>
</tr>
<tr>
<td>OperationTimeout</td>
<td>IdentityStoreConfig.CONN_TIMEOUT</td>
</tr>
<tr>
<td>ConnectTimeout</td>
<td>IdentityStoreConfig.CONNECT_TIMEOUT</td>
</tr>
<tr>
<td>HeartbeatInterval</td>
<td>IdentityStoreConfig.HEARTBEAT_INTERVAL</td>
</tr>
<tr>
<td>SocketOptions</td>
<td>IdentityStoreConfig.SOCKEET_READTIMEOUT</td>
</tr>
<tr>
<td></td>
<td>IdentityStoreConfig.SOCKEET_REUSEADDRESS</td>
</tr>
<tr>
<td></td>
<td>IdentityStoreConfig.SOCKEET_KEEPALIVE</td>
</tr>
<tr>
<td></td>
<td>IdentityStoreConfig.SOCKEET_TCPNODELAY</td>
</tr>
<tr>
<td>MaxPoolConnectionReuseTime</td>
<td>IdentityStoreConfig.MAX_POOL_CONNECTION_REUSE_TIME</td>
</tr>
<tr>
<td>PoolConnectionReclaimTime</td>
<td>IdentityStoreConfig.POLL_CONNECTION_RECLAIM_TIME</td>
</tr>
</tbody>
</table>

4.5.4 Handling Firewall and Load Balancer Timeout Errors

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.