Oracle® Fusion Middleware

Developing Applications with Identity Directory Services

14*c* (14.1.2.1.0)
G10435-01
March 2025

Oracle Fusion Middleware Developing Applications with Identity Directory Services, 14*c* (14.1.2.1.0)

G10435-01

Primary Author: Devanshi Mohan

Contents

Preface

This guide provides an introduction to Identity Directory Services 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 Directory 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:

- *Securing Applications with Oracle Platform Security Services*
- *Oracle® Fusion Middleware Infrastructure Security WLST Command Reference*

Conventions

The following text conventions are used in this document:

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

What's New in This Guide

Review the new features and significant product changes for the Identity Directory Services (IDS) and the related developer APIs.

New Features in Release 14c (14.1.2.1.0)

The new features and major enhancements introduced in release 14.1.2.1.0 include:

- **JDK Upgrade:** Identity Directory Services 14c (14.1.2.1.0) is certified for use with JDK 17, which introduces new features, optimizations, and bug fixes enhancing the overall performance and stability.
- **TLS 1.3 Support Added:** Support for TLS 1.3 and its ciphers has been integrated into the IDS layer, enhancing security and compliance with the latest TLS standards.
- **Deprecating TLS 1.1:** Support for TLS 1.1 and earlier versions has been discontinued. Identity Directory Services now requires TLS 1.2 or TLS 1.3 for secure communication.

1 Introduction to Identity Directory Services

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

- Overview of Identity Directory Services
- Understanding Identity Directory Services APIs
- System Requirements and Certification for Identity Directory Services

1.1 Overview of Identity Directory Services

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

Identity Directory Services is designed to meet the following goal:

- Simplify the development of identity information access regardless of where that information is stored.

The specifications provide a common framework for defining usage policies, attribute requirements, and developer APIs pertaining to the use of identity-related information. These enable businesses to ensure full documentation, control, and auditing regarding the use, storage, and propagation of identity-related data across systems and applications.

This section contains the following topics:

- Benefits of Identity Directory Services to Organizations
- Benefits of Identity Directory Services to Developers

1.1.1 Benefits of Identity Directory Services to Organizations

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

Organizations need to maintain control and integrity of sensitive personal information about their customers, employees, and partners. Data related to social security numbers, credit card numbers, medical history and more are increasingly under scrutiny by regulations seeking to prevent abuse or theft of such information. Privacy conscious organizations frequently have reacted to these requirements by enforcing overly strict controls and processes that hinder business operations and impact productivity, flexibility, and efficiency. At the opposite end of the spectrum, some organizations do not take the care needed to safeguard this information, potentially putting identity-related data at risk without sufficient oversight and control. The Identity Directory Services 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 Directory Services to Developers

The Identity Directory Services is an agreed-upon process for specifying how identity-related data is treated when writing applications. This provides developers a standard approach to write applications that use this data so that governing policies can be used to control it. This results in faster development of privacy aware applications.

IDS enables the decoupling of identity-aware applications from a specific deployment infrastructure. Specifically, using IDS 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. See Using the Identity Directory API.

1.2 Understanding Identity Directory Services APIs

Identity Directory Services rely on the Identity Directory API.

The following API is available based on Identity Directory Services:

- **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. See Using the Identity Directory API.

  Note:

  The Identity Directory API Object should be initialized only once, as it internally starts the full IDS stack for further initialization. Initializing multiple Identity Directory API Objects can create performance and stability bottlenecks for the application. In addition, you must ensure that the Identity Directory API Object is closed after its usage is complete.

1.3 System Requirements and Certification for Identity Directory Services

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

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

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

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

2

Using the Identity Directory API

The Identity Directory API supports accessing and managing users, groups, organizations, and can be extended to support new entity types with relationships defined between these entities.

> **Note:**
>
> Oracle recommends that you use the Identity Directory API for aggregating identity information from single data source only. You must keep in mind that the API does not support data aggregation from multiple data sources.

The following topics describe the architecture and key functionality of the Identity Directory Service.

2.1 Overview of the Identity Directory API

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

The API can be used in both Java EE and Java SE modes and 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.

The IDS API operates in SSL mode using the DefaultAuthenticator (Embedded LDAP).

Identity Directory Service consists of the following:

- **Identity Directory API**

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

- **Identity Directory API Configuration**

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

This section contains the following topics:

- Understanding Identity Directory API
- Identity Directory Service Architecture

2.1.1 Understanding Identity Directory API

The Identity Directory Service is a common service used by identity management products to access and manage an Identity Directory. The Identity Directory API is used to initialize the Identity Directory Service.

The Identity Directory Service provides an interface to both access and modify users and group information from different identity stores. An Identity Directory is an instance of the Identity Directory Service having:

- a unique name (IDS name)
- a logical entity configuration
- a physical identity store configuration

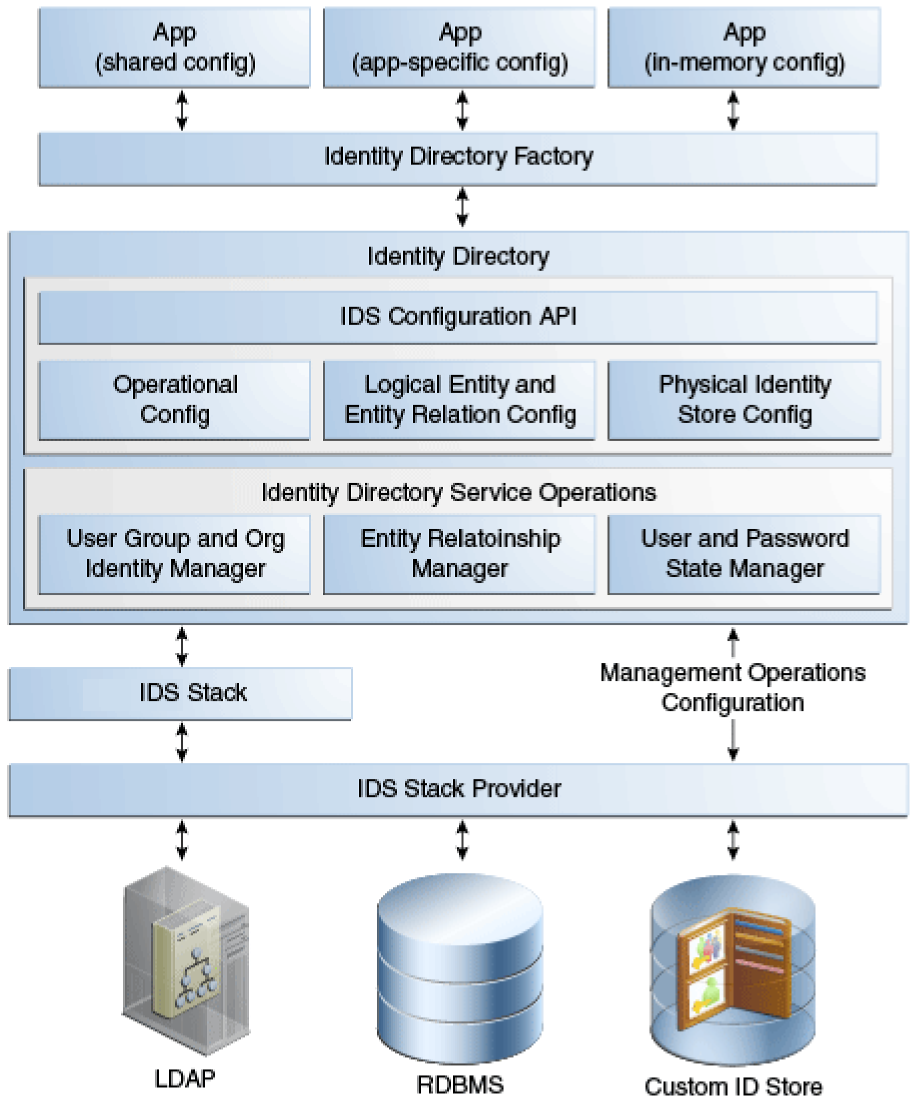
For more information about the Identity Directory Service, also referred to as the Identity Store Service, see Introduction to the Identity Store Service in *Oracle® Fusion Middleware Securing Applications with Oracle Platform Security Services*.

2.1.2 Identity Directory Service Architecture

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

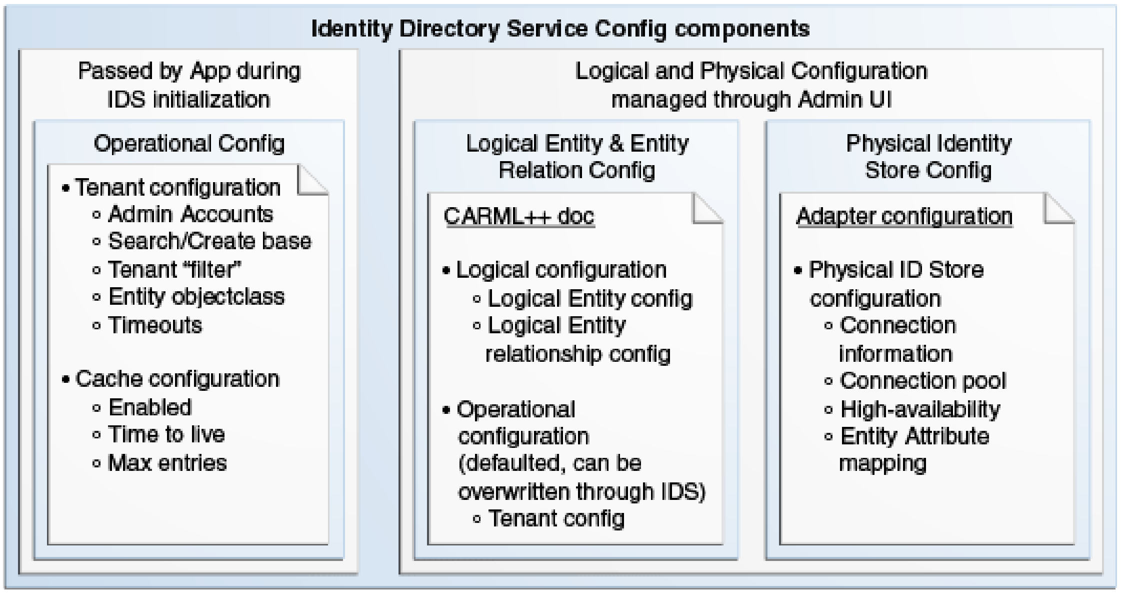
The following illustration shows the logical architecture of the Identity Directory Service.

Figure 2-1 Identity Directory API Architecture


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

The following illustration shows the relationship between the Identity Directory Service components.

Figure 2-2   Identity Directory API Components


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

2.2 Identity Directory API Configuration

The Identify Directory API provides an interface to access and modify users and group information from different identity stores. It is a combination of the logical entity configuration, the physical identity store configuration, and the operational configuration.

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

DOMAIN_HOME/config/fmwconfig/ids-config.xml

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

DOMAIN_HOME/config/fmwconfig

This section contains the following topics:

- Logical Entity Configuration for an Identity Directory Service
- Physical Identity Store Configuration for an Identity Directory Service
- Operational Configuration for an Identity Directory Service

2.2.1 Logical Entity Configuration for an Identity Directory Service

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

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

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

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

2.2.1.1 Properties of a Logical Entity Configuration

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

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

2.2.1.2 Attributes of a Logical Entity Configuration

The following table describes the logical entity attributes:

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

Note:

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

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

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

2.2.1.3 Properties of a Logical Entity Definition

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

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

2.2.1.4 Properties of a Logical Entity Relationship

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

Name	Description
`name`	Name of the entity relationship.
`type`	Valid entity values are as follows: `OneToOne`, `OneToMany`, `ManyToOne`, and `ManyToMany`.
`fromEntity`	Name of the first entity in the Entity Relationship.
`fromAttr`	The first entity's attribute. Value of this attribute relates to the second entity in the relationship.
`toEntity`	Name of the second entity in the Entity Relationship.
`toAttr`	The second entity's attribute. Value of the `fromAttr` property maps to this attribute in second entity.
`recursive`	Use `true` if the entity relationship is recursive. Default is `false`.

2.2.2 Physical Identity Store Configuration for an Identity Directory Service

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

The following table describes the physical identity store configuration properties:

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

2.2.3 Operational Configuration for an Identity Directory Service

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

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

The following table describes the operational configuration entities:

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

2.3 Design Recommendations for Identity Directory API

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

The following topics describe these recommendations:

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

2.3.1 Minimizing Use of defaultFetch Attributes

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

The entity attribute is defined by `defaultFetch` value. Also, try to have large attributes like `jpegphoto` configured with a `defaultFetch` value of false. The reason is every time the entity is read from the backend, all the `defaultFetch` attributes from backend directory will be retrieved. Too many `defaultFetch` attributes will affect the performance.

2.3.2 Initializing the Identity Directory Once

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

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

2.4 Examples of Using the Identity Directory API

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

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

- Initializing and Obtaining Identity Directory Handle
- Initializing and Obtaining Identity Directory Handle from JPS Context
- Initializing and Obtaining In-Memory Identity Directory Handle
- Adding a User
- Obtaining a User for Given Principal
- Modifying a User
- Obtaining a User for Given ID Value
- Searching Users Using Complex Search Filter
- Changing User Password
- Resetting User Password
- Authenticating a User
- Deleting a User
- Creating a Group
- Searching Groups
- Obtaining Management Chain
- Obtaining Reportees of a User
- Adding a Member to a Group
- Deleting a Member From a Group
- Obtaining All The Groups For Which User is a Member
- Using Logical NOT Operator in a Search Filter

2.4.1 Initializing and Obtaining Identity Directory Handle

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

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


public class IdsSample {

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

    public IdsSample() throws IDSException {
        // Set Operational Config
        OperationalConfig opConfig = new OperationalConfig();

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

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

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

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

Note:

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

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

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

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

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

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

2.4.2 Initializing and Obtaining Identity Directory Handle from JPS Context

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

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

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

public class IdsSample {

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

    public IdsSample() throws IDSException {

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

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

2.4.3 Initializing and Obtaining In-Memory Identity Directory Handle

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

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

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

public class IdsSample {

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

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

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

        // Add Group entity definition
        EntityDef groupEntityDef = new EntityDef("Group",
 EntityDef.EntityType.GROUP, "cn");
        groupEntityDef.addAttribute(new AttributeRef("cn"));
        groupEntityDef.addAttribute(new AttributeRef("uniquemember", false,
 AttributeRef.FilterType.EQUALS));
        entityDefs.add(groupEntityDef);
        // Add Entity relationship definition
        List<EntityRelationship> entityRelations = new
ArrayList<EntityRelationship>();
        entityRelations.add(new EntityRelationship("user_memberOfGroup",
                EntityRelationship.RelationshipType.MANYTOMANY, "User",
"principal", "Group", "uniquemember"));
        entityRelations.add(new EntityRelationship("group_memberOfGroup",
                EntityRelationship.RelationshipType.MANYTOMANY, "Group",
"principal", "Group", "uniquemember", true));
        EntitiesConfig entityCfg = new EntitiesConfig(attrDefs,
entityDefs, entityRelations);


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

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

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

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

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

```
}
```

2.4.4 Adding a User

After obtaining the Identity Directory handle, you can perform CRUD operations on users and groups. Use the sample code to add a user to the identity store.

```
Principal principal = null;

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

        try {
            CreateOptions createOpts = new CreateOptions();

            principal = uMgr.createUser(attrs, createOpts);



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

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

2.4.5 Obtaining a User for Given Principal

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

```
User user = null;

        try {
            ReadOptions readOpts = new ReadOptions();

            user = uMgr.getUser(principal, readOpts);

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

2.4.6 Modifying a User

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

```
try {
            ModifyOptions modifyOpts = new ModifyOptions();

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


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

2.4.7 Obtaining a User for Given ID Value

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

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

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

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

2.4.8 Searching Users Using Complex Search Filter

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

```
try {
            // Complex search filter with nested AND and OR conditiions
            SearchFilter filter = new SearchFilter(
                SearchFilter.LogicalOp.OR,
                new SearchFilter(SearchFilter.LogicalOp.AND,
                  new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH,
 "test"),
                  new SearchFilter("telephone", SearchFilter.Operator.CONTAINS,
 "650")),
                new SearchFilter(SearchFilter.LogicalOp.AND,
                  new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH,
 "demo"),
                  new SearchFilter(SearchFilter.LogicalOp.OR,
                    new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH,
 "hr"),
                    new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH,
 "it"),

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

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

            SearchOptions searchOpts = new SearchOptions();
            searchOpts.setPageSize(100);
            searchOpts.setRequestedAttrs(reqAttrs);
            searchOpts.setSortAttrs(new String[] {"firstname"});
            ResultSet<User> sr = uMgr.searchUsers(filter, searchOpts);
            while (sr.hasMore()) {
                User user = sr.getNext();
                System.out.println(user.getSubjectName());
            }

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

2.4.9 Changing User Password

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

```
ModifyOptions modOpts = new ModifyOptions();

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

2.4.10 Resetting User Password

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

```
ModifyOptions modOpts = new ModifyOptions();

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

2.4.11 Authenticating a User

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

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

2.4.12 Deleting a User

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

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

            uMgr.deleteUser(principal, deleteOpts);

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

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

2.4.13 Creating a Group

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

```
Principal principal = null;

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

            principal = gMgr.createGroup(attrs, createOpts);

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

2.4.14 Searching Groups

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

```
public void searchGroups() {

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

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

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

2.4.15 Obtaining Management Chain

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

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

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

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

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

2.4.16 Obtaining Reportees of a User

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

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

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

            SearchOptions searchOpts = new SearchOptions();
            searchOpts.setPageSize(20);
            int nLevels = 0;

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

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

2.4.17 Adding a Member to a Group

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

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

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

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

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

2.4.18 Deleting a Member From a Group

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

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

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

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

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

Note:

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

2.4.19 Obtaining All The Groups For Which User is a Member

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

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

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

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

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

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

2.4.20 Using Logical NOT Operator in a Search Filter

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

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

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

2.5 Supported Cipher Suites in Identity Directory Services

Learn about the cipher suites that Identity Directory Services uses.

This section contains the following topics.

- Supported Cipher Suites for Identity Directory Services in AIX
- Adding Supported Cipher Suites in adapters.os_xml

2.5.1 Supported Cipher Suites for Identity Directory Services in AIX

This section provides a list of the cipher suites supported by the IBM JDK, which are enabled by default.

Note:

It is recommended to avoid using weak ciphers for IDS in AIX. While IDS does not offer additional cipher support, it will still communicate with the backend server using the specified ciphers. Therefore, it's advisable to avoid weak ciphers and remove them from the LDAP backend server as well.

```
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
    SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
    SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384
```

```
SSL_RSA_WITH_AES_256_CBC_SHA256
SSL_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
SSL_ECDH_RSA_WITH_AES_256_CBC_SHA384
SSL_DHE_RSA_WITH_AES_256_CBC_SHA256
SSL_DHE_DSS_WITH_AES_256_CBC_SHA256
SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA
SSL_RSA_WITH_AES_256_CBC_SHA
SSL_ECDH_ECDSA_WITH_AES_256_CBC_SHA
SSL_ECDH_RSA_WITH_AES_256_CBC_SHA
SSL_DHE_RSA_WITH_AES_256_CBC_SHA
SSL_DHE_DSS_WITH_AES_256_CBC_SHA
SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256
SSL_RSA_WITH_AES_128_CBC_SHA256
SSL_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
SSL_ECDH_RSA_WITH_AES_128_CBC_SHA256
SSL_DHE_RSA_WITH_AES_128_CBC_SHA256
SSL_DHE_DSS_WITH_AES_128_CBC_SHA256
SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_AES_128_CBC_SHA
SSL_ECDH_ECDSA_WITH_AES_128_CBC_SHA
SSL_ECDH_RSA_WITH_AES_128_CBC_SHA
SSL_DHE_RSA_WITH_AES_128_CBC_SHA
SSL_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384
SSL_RSA_WITH_AES_256_GCM_SHA384
SSL_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
SSL_ECDH_RSA_WITH_AES_256_GCM_SHA384
SSL_DHE_DSS_WITH_AES_256_GCM_SHA384
SSL_DHE_RSA_WITH_AES_256_GCM_SHA384
SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256
SSL_RSA_WITH_AES_128_GCM_SHA256
SSL_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
SSL_ECDH_RSA_WITH_AES_128_GCM_SHA256
SSL_DHE_RSA_WITH_AES_128_GCM_SHA256
SSL_DHE_DSS_WITH_AES_128_GCM_SHA256
```

2.5.2 Adding Supported Cipher Suites in adapters.os_xml

You can add IBM JDK enabled ciphers in adapters.os_xml.

To add the ciphers:

1. Open the adapters.os_xml file.
2. Add the required ciphers in adapters.os_xml as shown below:

```
<ldap id="DefaultAuthenticator" version="0">
    ...
    <ssl>
    <protocols>TLSv1.2,TLSv1.1</protocols>
    <cipherSuites>
    <cipher>SSL_RSA_WITH_AES_128_CBC_SHA</cipher>
    <cipher>SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA</cipher>

    ...

    <cipher>SSL_ECDH_ECDSA_WITH_AES_128_GCM_SHA256</cipher>
    </cipherSuites>
```

```
</ssl>
...
</ldap>
```

3. Restart the weblogic server.