Oracle® Fusion Middleware

Application Developer's Guide for Oracle Identity Management

ORACLE®

Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management, 12*c* (12.2.1.4.0)

E95868-04

The *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management* explains how to modify applications to work with Oracle Identity Management, including Oracle Application Server Single Sign-On, Oracle Internet Directory, Oracle Delegated Administration Services, and the Directory Integration Platform.

This preface contains these topics:

Contents

3   Extensions to the LDAP Protocol

4 Developing Applications With Oracle Extensions to the Standard APIs

5   Using the Java API Extensions to JNDI

6   Using the API Extensions in PL/SQL

7   Developing Provisioning-Integrated Applications

Part II   Oracle Internet Directory Programming Reference

8   C API Reference

9   DBMS_LDAP PL/SQL Reference

10   Java API Reference

11   DBMS_LDAP_UTL PL/SQL Reference

12   Oracle Directory Integration and Provisioning Java API Reference

13   Oracle Directory Integration Platform PL/SQL API Reference

Part III Appendixes

A Java Plug-ins for User Provisioning

B DSML Syntax

C Migrating from Netscape LDAP SDK API to Oracle LDAP SDK API

List of Figures

List of Tables

Preface

The *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management* explains how to modify applications to work with Oracle Identity Management, including Oracle Application Server Single Sign-On, Oracle Internet Directory, Oracle Delegated Administration Services, and the Directory Integration Platform.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

The following readers can benefit from this book:

- Developers who want to integrate applications with Oracle Identity Management. This process involves storing and updating information in an Oracle Internet Directory server. It also involves modifying applications to work with mod_osso, an authentication module on the Oracle HTTP Server.
- Anyone who wants to learn about the LDAP APIs and Oracle extensions to these 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 these Oracle resources:

- *Oracle Fusion Middleware Installation Guide for Oracle Identity Management*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*

- *Oracle® Fusion Middleware Application Developer's Guide for Oracle Identity Management*

For additional information, see:

- Chadwick, David. *Understanding X.500—The Directory.* Thomson Computer Press, 1996.

- Howes, Tim and Mark Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol.* Macmillan Technical Publishing, 1997.

- Howes, Tim, Mark Smith and Gordon Good, *Understanding and Deploying LDAP Directory Services.* Macmillan Technical Publishing, 1999.

- Internet Assigned Numbers Authority home page, `http://www.iana.org`, for information about object identifiers

- Internet Engineering Task Force (IETF) documentation available at: `http://www.ietf.org`, especially:

  – The LDAPEXT charter and LDAP drafts

  – The LDUP charter and drafts

  – RFC 2251, "Lightweight Directory Access Protocol (v3)"

  – RFC 2254, "The String Representation of LDAP Search Filters"

  – RFC 1823, "The LDAP Application Program Interface"

- The OpenLDAP Community, `http://www.openldap.org`

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.

Preface

The *Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management* explains how to modify applications to work with Oracle Identity Management, including Oracle Application Server Single Sign-On, Oracle Internet Directory, Oracle Delegated Administration Services, and the Directory Integration Platform.

This preface contains these topics:

- Audience

- Documentation Accessibility

- Related Documents

- Conventions

Audience

The following readers can benefit from this book:

- Developers who want to integrate applications with Oracle Identity Management. This process involves storing and updating information in an Oracle Internet Directory server. It also involves modifying applications to work with mod_osso, an authentication module on the Oracle HTTP Server.

- Anyone who wants to learn about the LDAP APIs and Oracle extensions to these 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 these Oracle resources:

- *Oracle Fusion Middleware Installation Guide for Oracle Identity Management*

- *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*

- *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*

- *Oracle® Fusion Middleware Application Developer's Guide for Oracle Identity Management*

For additional information, see:

- Chadwick, David. *Understanding X.500—The Directory.* Thomson Computer Press, 1996.

- Howes, Tim and Mark Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol.* Macmillan Technical Publishing, 1997.

- Howes, Tim, Mark Smith and Gordon Good, *Understanding and Deploying LDAP Directory Services.* Macmillan Technical Publishing, 1999.

- Internet Assigned Numbers Authority home page, `http://www.iana.org`, for information about object identifiers

- Internet Engineering Task Force (IETF) documentation available at: `http://www.ietf.org`, especially:

  - The LDAPEXT charter and LDAP drafts

  - The LDUP charter and drafts

  - RFC 2251, "Lightweight Directory Access Protocol (v3)"

    – RFC 2254, "The String Representation of LDAP Search Filters"

    – RFC 1823, "The LDAP Application Program Interface"

- The OpenLDAP Community, http://www.openldap.org

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 the SDK?

This chapter describes the new features of the Software Developer's Kit (SDK) for Oracle Identity Management.

This chapter includes the following sections:

- New Features in the 12*c* (12.2.1.4.0) SDK
- New Features in the 12*c* (12.2.1.3.0) SDK

New Features in the 12*c* (12.2.1.4.0) SDK

This revision contains no new features.

For a comprehensive listing of the new Oracle Internet Directory features introduced in this release, see New and Changed Features for Oracle Internet Directory 12c (12.2.1.4.0) in *Administering Oracle Internet Directory*.

New Features in the 12*c* (12.2.1.3.0) SDK

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

For a comprehensive listing of the new Oracle Internet Directory features introduced in this release, see New and Changed Features for Oracle Internet Directory 12c (12.2.1.3.0) in *Administering Oracle Internet Directory*.

Part I

Programming for Oracle Identity Management

You can modify your applications to work with the different components of Oracle Identity Management. This section begins with an introduction to the Oracle Internet Directory SDK and to LDAP programming concepts. You then learn how to use the three LDAP APIs and their extensions to enable applications for Oracle Internet Directory.

Part I contains these chapters:

- Developing Applications for Oracle Identity Management
- Developing Applications with Standard LDAP APIs
- Extensions to the LDAP Protocol
- Developing Applications With Oracle Extensions to the Standard APIs
- Using the Java API Extensions to JNDI
- Using the API Extensions in PL/SQL
- Developing Provisioning-Integrated Applications

1

Developing Applications for Oracle Identity Management

Oracle Identity Management provides a shared infrastructure for all Oracle applications. It also provides services and interfaces that facilitate third-party enterprise application development. These interfaces are useful for application developers who need to incorporate identity management into their applications.
As of Release 11g Release 1 (11.1.1), the recommended security API for Fusion Middleware application developers is Oracle Platform Security for Java, which is documented in the *Fusion Middleware Security Guide*. The Oracle Identity Management interfaces described in the current book are not part of Oracle Platform Security for Java.

The following topics describe the interfaces for developing applications for Oracle Identity Management and recommends application development best practices in the Oracle Identity Management environment.:

- Overview of Oracle Identity Management Services Available for Application Integration
- Understanding Existing Applications Integration with Oracle Identity Management
- Overview of Oracle Identity Management Programming

1.1 Overview of Oracle Identity Management Services Available for Application Integration

Custom applications such as Oracle Internet Directory, Oracle Delegated Administration Services and Oracle Provisioning Integration Services can use Oracle Identity Management through a set of documented and supported services and APIs.

For example:

- Oracle Internet Directory provides LDAP APIs for C, Java, and PL/SQL, and is compatible with other LDAP SDKs.
- Oracle Delegated Administration Services provides a core self-service console that can be customized to support third-party applications. In addition, they provide several services for building customized administration interfaces that manipulate directory data.
- Oracle Directory Integration Services facilitate the development and deployment of custom solutions for synchronizing Oracle Internet Directory with third-party directories and other user repositories.
- Oracle Provisioning Integration Services provide a mechanism for provisioning third-party applications, and a means of integrating the Oracle environment with other provisioning systems.
- Oracle Single Sign-On provides APIs for developing and deploying partner applications that share a single sign-on session with other Oracle Web applications.
- JAZN is the Oracle implementation of the Java Authentication and Authorization Service (JAAS) Support standard. JAZN allows applications developed for the Web using the

Oracle J2EE environment to use the identity management infrastructure for authentication and authorization.

1.2 Understanding Existing Applications Integration with Oracle Identity Management

An enterprise deploys certain applications to perform critical business functions. Oracle Identity Management provides services that can be leveraged by the deployment to modify existing applications.

For new applications, use Oracle Platform Security for Java, which is documented in the *Fusion Middleware Security Guide*.

Oracle Identity Management provides the following services that can be leveraged by the deployment to modify existing applications:

The services that can be leveraged by Oracle Identity Management are:

- **Automated User Provisioning**: The deployment can develop a custom provisioning agent that automates the provisioning of users in the existing application in response to provisioning events in the Oracle Identity Management infrastructure. This agent must be developed using the interfaces of Oracle Provisioning Integration Service.

> ✎ **See Also:**
>
> Introduction to Managing Directory Entries in Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory for more information about developing automated user provisioning.

- **User Authentication Services**: If the user interface of the existing application is based on HTTP, integrating it with Oracle HTTP Server and protecting its URL using `mod_osso` authenticates all incoming user requests using the service.

- **Centralized User Profile Management**: If the user interface of the existing application is based on HTTP, and it is integrated with Oracle Single Sign-On for authentication, the application can use the self-service console of Oracle Delegated Administration Services to enable centralized user profile management. The self-service console can be customized by the deployment to address the specific needs of the application.

1.3 Overview of Oracle Identity Management Programming

Oracle Identity Management Software Developer's Kit provides an overview of how an application can use the kit to integrate with the directory. You are also acquainted with the rest of the directory product suite.

The following topics describe Oracle Identity Management Programming:

- About Programming Languages Supported by Oracle Internet Directory SDK
- About Oracle Identity Management SDK Components
- Developing Application in the Oracle Identity Management Environment

1.3.1 About Programming Languages Supported by Oracle Internet Directory SDK

Oracle Internet Directory SDK supports programming languages.

The SDK is for application developers who use C, C++, and PL/SQL. Java developers must use the JNDI provider to integrate with the directory.

1.3.2 About Oracle Identity Management SDK Components

The Oracle Identity Management Software Developer's Kit consists components to integrate with the directory.

- A C API compliant with LDAP Version 3
- A PL/SQL API contained in a PL/SQL package called `DBMS_LDAP`
- *Oracle Identity Management Application Developer's Guide* (this document)
- Command-line tools

1.3.3 Developing Application in the Oracle Identity Management Environment

Oracle Identity Management applications are back-end programs that handle multiple requests from multiple users. You need to create an application identity in the directory to develop an application.

- Oracle Identity Management Application Architecture
- Oracle Identity Management Interactions During the Application Life Cycle
- Services and APIs to Integrate Applications with Oracle Identity Management
- Example of Using APIs and Services for Application Integration with Oracle Internet Directory
- Services to Integrate Existing Applications with Oracle Identity Management

1.3.3.1 Oracle Identity Management Application Architecture

Most Oracle Identity Management applications are back-end programs that simultaneously handle multiple requests from multiple users. Figure 1-1 shows how a directory is used by such applications.

Figure 1-1   A Directory-Enabled Application
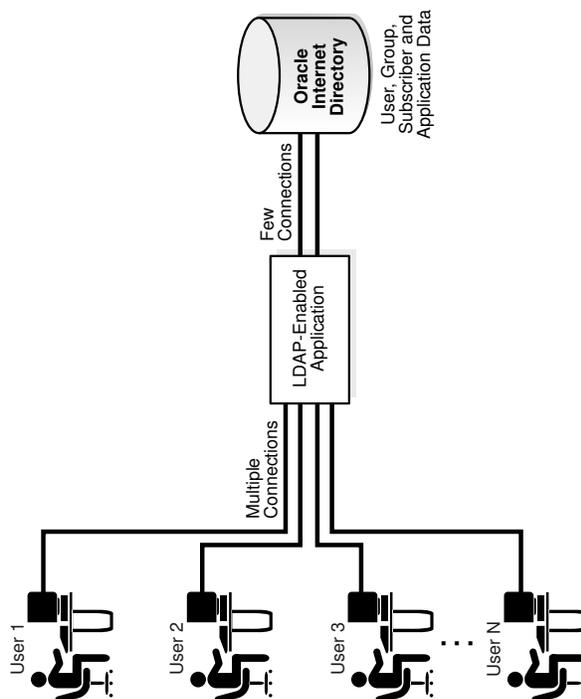
As Figure 1-1 shows, when a user request involves an LDAP-enabled operation, the application processes the request using a smaller set of pre-created directory connections.

1.3.3.2 Oracle Identity Management Interactions During the Application Life Cycle

Table 1-1 walks you through the directory operations that an application typically performs during its lifecycle.

Table 1-1   Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Installation	Do the following: • Create an application identity in the directory. The application uses this identity to perform most of its LDAP operations. • Give the application identity LDAP authorizations by making it part of the correct LDAP groups. These authorizations enable the application to accept user credentials and authenticate them against the directory. The directory can also use application authorizations to proxy for the user when LDAP operations must be performed on the user's behalf.

Table 1-1   (Cont.) Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Startup and Bootstrap	The application must retrieve credentials that enable it to authenticate itself to the directory.
	If the application stores configuration metadata in Oracle Internet Directory, it can retrieve that metadata and initialize other parts of the application.
	The application can then establish a pool of connections to serve user requests.
Application Runtime	For every end-user request that needs an LDAP operation, the application can:
	• Pick a connection from the pool of LDAP connections.
	• Switch the user to the end-user identity if the LDAP operation must be performed with the effective rights of the end-user.
	• Perform the LDAP operation by using either the regular API or the API enhancements described in this chapter.
	• Ensure that the effective user is now the application identity when the LDAP operation is complete.
	• Return the LDAP connection back to the pool of connections.
Application Shutdown	Abandon any outstanding LDAP operations and close all LDAP connections.
Application Deinstallation	Remove the application identity and the LDAP authorizations granted to it.

1.3.3.3 Services and APIs to Integrate Applications with Oracle Identity Management

Application developers can integrate with Oracle Identity Management by using the services and APIs listed described in Table 1-2.

Table 1-2   Services and APIs for Integrating with Oracle Internet Directory

Service/API	Description	More Information
Standard LDAP APIs in C, PL/SQL and Java	These provide basic LDAP operations. The standard LDAP API used in Java is the JNDI API with the LDAP service provider.	Developing Applications with Standard LDAP APIs
Oracle Extensions to Standard C, PL/SQL and Java APIs	These APIs provide programmatic interfaces that model various concepts related to identity management.	Developing Applications With Oracle Extensions to the Standard APIs

Table 1-2   (Cont.) Services and APIs for Integrating with Oracle Internet Directory

Service/API	Description	More Information
Oracle Delegated Administration Services	Oracle Delegated Administration Services consists of a self-service console and administrative interfaces. You can modify the administrative interfaces to support third-party applications.	The 10g (10.1.4.0.1) Library.
Oracle Directory Provisioning Integration Service	You can use the Oracle Provisioning Integration System to provision third-party applications and integrate other provisioning systems.	• Developing Provisioning-Integrated Applications • Deploying Provisioning-Integrated Applications in *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*

1.3.3.4 Example of Using APIs and Services for Application Integration with Oracle Internet Directory

Figure 1-2 shows an application leveraging some of the services listed in Table 1-2.

Figure 1-2   An Application Leveraging APIs and Services
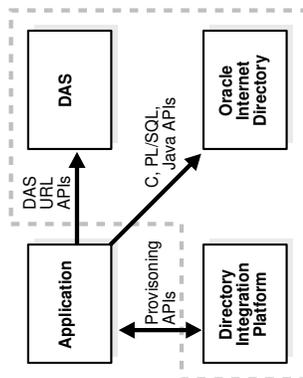
As Figure 1-2 shows, the application integrates with Oracle Internet Directory as follows:

• Using PL/SQL, C, or Java APIs, it performs LDAP operations directly against the directory.

• In some cases, it directs users to self-service features of Oracle Delegated Administration Services.

• It is notified of changes to entries for users or groups in Oracle Internet Directory. The Oracle Directory Provisioning Integration Service provides this notification.

1.3.3.5 Services to Integrate Existing Applications with Oracle Identity Management

Your enterprise may already have deployed applications that you may have wanted to integrate with the Oracle identity management infrastructure. You can integrate these applications using the services presented in Table 1-3.

Table 1-3   Services for Modifying Existing Applications

Service	Description	More Information
Automated User Provisioning	You can develop an agent that automatically provisions users when provisioning events occur in the Oracle identity management infrastructure. You use interfaces of the Oracle Directory Provisioning Integration Service to develop this agent.	Developing Provisioning-Integrated Applications
User Authentication Services	If your user interface is based on HTTP, you can integrate it with the Oracle HTTP Server. This enables you to use mod_osso and OracleAS Single Sign-On to protect the application URL.	*Oracle Application Server Single Sign-On Administrator's Guide*
Centralized User Profile Management	If your user interface is based on HTTP and is integrated with OracleAS Single Sign-On, you can use the Oracle Enterprise Single Sign-On Administrative Console to manage user profiles centrally. You can tailor the console to the needs of your application.	• The 10g (10.1.4.0.1) library. • The chapter about the delegated administration services framework in *Oracle Identity Management Guide to Delegated Administration*.

2

Developing Applications with Standard LDAP APIs

The standard LDAP API enables you to integrate your applications with the API. It is used to initialise, authenticate and terminate the sessions.

- Overview of LDAP History
- Understanding LDAP Models
- Using Standard LDAP APIs in Oracle Internet Directory
- Initializing an LDAP Session
- Authenticating an LDAP Session
- Searching the Directory
- Terminating the Session

2.1 Overview of LDAP History

LDAP began as a lightweight front end to the X.500 Directory Access Protocol. LDAP formats are easy to represent data.

LDAP simplifies the X.500 Directory Access Protocol in the following ways:

- It uses TCP/IP connections. These are lightweight compared to the OSI communication stack required by X.500 implementations
- It eliminates little-used and redundant features of the X.500 Directory Access Protocol
- It uses simple formats to represent data elements. These formats are easier to process than the complicated and highly structured representations in X.500.
- It uses a simplified version of the X.500 encoding rules used to transport data over networks.

2.2 Understanding LDAP Models

LDAP uses four basic models to define its operations to be performed on directory entries.

LDAP models are described in the following sections:

- About the Naming Model
- About the Information Model
- About the Functional Model
- About the Security Model

2.2.1 About the Naming Model

The LDAP naming model enables directory information to be referenced and organized. Each entry in a directory is uniquely identified by a distinguished name (DN). The DN tells you exactly where an entry resides in the directory hierarchy. A directory information tree (DIT) is used to represent this hierarchy.

Figure 2-1 illustrates the relationship between a distinguished name and a directory information tree.

Figure 2-1   A Directory Information Tree
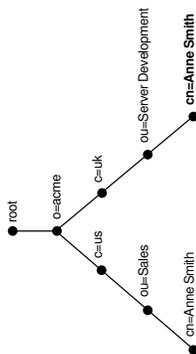
illustration: oidag025.eps
release: 9.2
caption: A Directory Information Tree
date: 3/11/02
platform: pc

root
c=acme
c=uk
ou=Server Development
cn=Anne Smith
c=us
ou=Sales
cn=Anne Smith

The DIT in Figure 2-1 shows entries for two employees of Example Corporation who are both named Anne Smith. It is structured along geographical and organizational lines. The Anne Smith represented by the left branch works in the Sales division in the United States. Her counterpart works in the Server Development division in the United Kingdom.

The Anne Smith represented by the right branch has the common name (`cn`) Anne Smith. She works in an organizational unit (`ou`) named Server Development, in the country (`c`) of United Kingdom of Great Britain and Northern Ireland (`uk`), in the organization (`o`) Example. The DN for this Anne Smith entry looks like this:

```
cn=Anne Smith,ou=Server Development,c=uk,o=example
```

Note that the conventional format for a distinguished name places the lowest DIT component at the left. The next highest component follows, on up to the root.

Within a distinguished name, the lowest component is called the relative distinguished name (RDN). In the DN just presented, the RDN is `cn=Anne Smith`. The RDN for the entry immediately above Anne Smith's RDN is `ou=Server Development`. And the RDN for the entry immediately above `ou=Server Development` is `c=uk`, and so on. A DN is thus a sequence of RDNs separated by commas.

To locate a particular entry within the overall DIT, a client uniquely identifies that entry by using the full DN—not simply the RDN—of that entry. To avoid confusion between the two Anne Smiths in the global organization depicted in Figure 2-1, you use the full DN for each. If there are two employees with the same name in the same organizational unit, you can use other mechanisms. You may, for example, use a unique identification number to identify these employees.

2.2.2 About the Information Model

The LDAP information model determines the form and character of information in the directory. This model uses the concept of entries as its defining characteristic. In a directory, an entry is a collection of information about an object. A telephone directory, for example, contains entries for people. A library card catalog contains entries for books. An online directory may contain entries for employees, conference rooms, e-commerce partners, or shared network resources such as printers.

In a typical telephone directory, a person entry contains an address and a phone number. In an online directory, each of these pieces of information is called an attribute. A typical employee entry contains attributes for a job title, an e-mail address, and a phone number.

In Figure 2-2, the entry for Anne Smith in Great Britain (uk) has several attributes. Each provides specific information about her. Those listed in the balloon to the right of the tree are `emailaddrs`, `printername`, `jpegPhoto`, and `app preferences`. Note that the rest of the bullets in Figure 2-2 are also entries with attributes, although these attributes are not shown.

Figure 2-2   Attributes of the Entry for Anne Smith
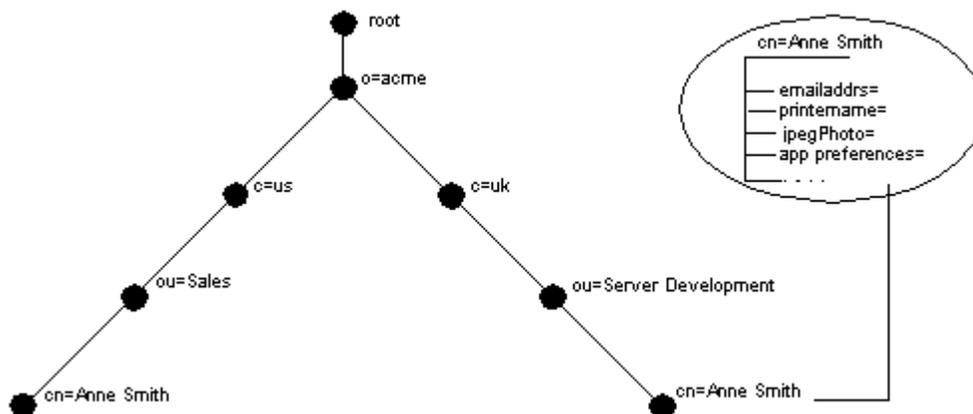
Each attribute consists of an attribute type and one or more attribute values. The attribute type is the kind of information that the attribute contains—jobTitle, for instance. The attribute value is the actual information. The value for the jobTitle attribute, for example, might be manager.

2.2.3 About the Functional Model

The LDAP functional model determines what operations can be performed on directory entries. There operations are mainly search, modify and authenticate.

Table 2-1 lists and describes the functions:

Table 2-1   LDAP Functions

Function	Description
Search and read	The read operation retrieves the attributes of an entry whose name is known. The list operation enumerates the children of a given entry. The search operation selects entries from a defined area of the tree based on some selection criteria known as a search filter. For each matching entry, a requested set of attributes (with or without values) is returned. The searched entries can span a single entry, an entry's children, or an entire subtree. Alias entries can be followed automatically during a search, even if they cross server boundaries. An abandon operation is also defined, allowing an operation in progress to be canceled.
Modify	This category defines four operations that modify the directory: • Modify—change existing entries. You can add and delete values. • Add—insert entries into the directory • Delete—remove entries from the directory • Modify RDN—change the name of an entry
Authenticate	This category defines a bind operation. A bind enables a client to initiate a session and prove its identity to the directory. Oracle Internet Directory supports several authentication methods, from simple clear-text passwords to public keys. The unbind operation is used to terminate a directory session.

2.2.4 About the Security Model

The LDAP security model enables directory information to be secured. It ensures the data security and integrity in Oracle Internet Directory.

This model has several parts as described in the following sections:

- Understanding Authentication Types in Oracle Internet Directory
- Understanding Access Control and Authorization in Oracle Internet Directory
- Understanding Data Integrity in Oracle Internet Directory
- Understanding Data Privacy in Oracle Internet Directory
- Understanding Password Policies in Oracle Internet Directory

2.2.4.1 Understanding Authentication Types in Oracle Internet Directory

Authentication is the process by which the directory server establishes the identity of the user connecting to the directory. Directory authentication occurs when an LDAP bind operation establishes an LDAP session. Every session has an associated user identity, also referred to as an authorization ID.

Authentication options of Oracle Internet Directory is described in the following sections:

- About Anonymous Authentication
- About Simple Authentication
- About Authentication Using Secure Sockets Layer (SSL)

2.2.4.1.1 About Anonymous Authentication

If your directory is available to everyone, users may log in anonymously. In anonymous authentication, users leave the user name and password fields blank when they log in. They then exercise whatever privileges are specified for anonymous users.

2.2.4.1.2 About Simple Authentication

In simple authentication, the client uses an unencrypted DN and password to identify itself to the server. The server verifies that the client's DN and password match the DN and password stored in the directory.

2.2.4.1.3 About Authentication Using Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections. It uses a certificate exchange to authenticate users. These certificates are verified by trusted certificate authorities. A certificate ensures that an entity's identity information is correct. An entity can be an end user, a database, an administrator, a client, or a server. A Certificate Authority (CA) is an application that creates public key certificates that are given a high level of trust by all parties involved.

You can use SSL in one of the three authentication modes presented in Table 2-2.

Table 2-2   SSL Authentication Modes

SSL Mode	Description
No authentication	Neither the client nor the server authenticates itself to the other. No certificates are sent or exchanged. In this case, only SSL encryption and decryption are used.
One-way authentication	Only the directory server authenticates itself to the client. The directory server sends the client a certificate verifying that the server is authentic.
Two-way authentication	Both client and server authenticate themselves to each other, exchanging certificates.

In an Oracle Internet Directory environment, SSL authentication between a client and a directory server involves three basic steps:

1. The user initiates an LDAP connection to the directory server by using SSL on an SSL port. The default SSL port is `3131`.

2. SSL performs the handshake between the client and the directory server.

3. If the handshake is successful, the directory server verifies that the user has the appropriate authorization to access the directory.

> **✎ See Also:**
>
> *Oracle Advanced Security Administrator's Guide* for more information about SSL.

2.2.4.2 Understanding Access Control and Authorization in Oracle Internet Directory

The authorization process ensures that a user reads or updates only the information for which he or she has privileges. The directory server ensures that the user— identified by the authorization ID associated with the session—has the requisite permissions to perform a given directory operation. Absent these permissions, the operation is disallowed.

The mechanism that the directory server uses to ensure that the proper authorizations are in place is called access control. And an access control item (ACI) is the directory metadata that captures the administrative policies relating to access control.

An ACI is stored in Oracle Internet Directory as user-modifiable operational attributes. Typically a whole list of these ACI attribute values is associated with a directory object. This list is called an access control list (ACL). The attribute values on that list govern the access policies for the directory object.

ACIs are stored as text strings in the directory. These strings must conform to a well-defined format. Each valid value of an ACI attribute represents a distinct access control policy. These individual policy components are referred to as ACI Directives or ACIs and their format is called the ACI Directive format.

Access control policies can be prescriptive: their security directives can be set to apply downward to all entries at lower positions in the directory information tree (DIT). The point from which an access control policy applies is called an access control policy point (ACP).

2.2.4.3 Understanding Data Integrity in Oracle Internet Directory

Oracle Internet Directory uses SSL to ensure that data is not modified, deleted, or replayed during transmission. This feature uses cryptographic checksums to generate a secure message digest. The checksums are created using either the MD5 algorithm or the Secure Hash Algorithm (SHA). The message digest is included in each network packet.

2.2.4.4 Understanding Data Privacy in Oracle Internet Directory

Oracle Internet Directory uses public key encryption over SSL to ensure that data is not disclosed during transmission. In public-key encryption, the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the recipient decrypts the message using his or her private key. The directory supports two levels of encryption:

Table 2-3   Encryption algorithms

Algorithm	Description
DES40	The DES40 algorithm, available internationally, is a DES variant in which the secret key is preprocessed to provide forty effective key bits. It is designed for use by customers outside the USA and Canada who want to use a DES-based encryption algorithm.
RC4_40	Oracle is licensed to export the RC4 data encryption algorithm with a 40-bit key size to virtually all destinations where Oracle products are available. This makes it possible for international corporations to safeguard their entire operations with fast cryptography.

2.2.4.5 Understanding Password Policies in Oracle Internet Directory

A password policy is a set of rules that govern how passwords are used. When a user attempts to bind to the directory, the directory server uses the password policy to ensure that the password provided meets the various requirements set in that policy.

When you establish a password policy, you set the following types of rules, to mention just a few:

- The maximum length of time a given password is valid
- The minimum number of characters a password must contain
- The ability of users to change their passwords

2.3 Using Standard LDAP APIs in Oracle Internet Directory

The standard LDAP APIs are available in C, PL/SQL, and Java. They enable you to perform the fundamental LDAP operations.

LDAP operations are described in " Understanding LDAP Models". The C and PL/SQL APIs are part of the directory SDK. The Java is part of the JNDI package. All three use TCP/IP connections. They are based on LDAP Version 3, and they support SSL connections to Oracle Internet Directory.

The following sections explain how to get started with the APIs:

- Steps to Follow to Invoke the LDAP API
- Getting Started with the C API
- Getting Started with the Java API
- Getting Started with the DBMS_LDAP Package

2.3.1 Steps to Follow to Invoke the LDAP API

Typically, to invoke the LDAP API, you need to initialize and authenticate the session.

An application uses the functions in the API in four steps:

1. Initialize the library and obtain an LDAP session handle.
2. Authenticate to the LDAP server if necessary.
3. Perform some LDAP operations and obtain results and errors, if any.
4. Close the session.

Figure 2-3 illustrates these steps.

Figure 2-3   Steps in Typical DBMS_LDAP Usage
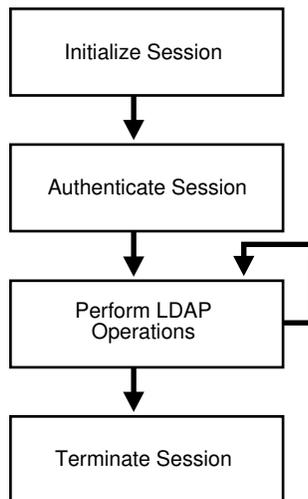
2.3.2 Getting Started with the C API

When you build applications with the C API, you must include the header file `ldap.h`, located at `$ORACLE_HOME/ldap/public`.

In addition, you must dynamically link to the library located at `$ORACLE_HOME/lib/libclntsh.so.10.1`.

> **✐ See Also:**
>
> Sample C API Usage to learn how to use the SSL and non-SSL modes.

2.3.3 Getting Started with the DBMS_LDAP Package

The `DBMS_LDAP` package enables PL/SQL applications to access data located in enterprise-wide LDAP servers. The names and syntax of the function calls are similar to those of the C API.

These functions comply with current recommendations of the Internet Engineering Task Force (IETF) for the C API. Note though that the PL/SQL API contains only a subset of the functions available in the C API. Most notably, only synchronous calls to the LDAP server are available in the PL/SQL API.

To begin using the PL/SQL LDAP API, use this command sequence to load DBMS_LDAP into the database:

1. Log in to the database, using SQL*Plus. Run the tool in the Oracle home in which your database is present. Connect as `SYSDBA`.

   ```
   SQL> CONNECT / AS SYSDBA
   ```

2. Load the API into the database, using this command:

   ```
   SQL> @?/rdbms/admin/catladap.sql
   ```

2.3.4 Getting Started with the Java API

Java developers can use the Java Naming and Directory Interface (JNDI) to gain access to information in Oracle Internet Directory.

The JNDI is found at this link:

http://www.oracle.com/technetwork/java/jndi/index.html

Although no Java APIs are provided in this chapter, Initializing the Session by Using JNDI shows you how to use wrapper methods for the JNDI to establish a basic connection.

2.4 Initializing an LDAP Session

All LDAP operations based on the C API require clients to establish an LDAP session with the LDAP server. For LDAP operations based on the PL/SQL API, a database session must first initialize and open an LDAP session.

Most Java operations require a Java Naming and Directory Interface (JNDI) connection. The `oracle.ldap.util.jndi` package, provided here, simplifies the work involved in achieving this connection.

LDAP session initialization is described in the following topics:

• Initializing the Session by Using the C API

• Initializing the Session by Using DBMS_LDAP

- Initializing the Session by Using JNDI

2.4.1 Initializing the Session by Using the C API

The C function `ldap_init()` initializes a session with an LDAP server. The server is not actually contacted until an operation is performed that requires it, allowing options to be set after initialization.

`ldap_init` has the following syntax:

```
LDAP *ldap_init
(

const char      *hostname,
int             portno

);
```

Table 2-4 lists and defines the function parameters.

Table 2-4   Parameters for ldap_init()

Parameter	Description
hostname	Contains a space-separated list of directory host names or IP addresses represented by dotted strings. You can pair each host name with a port number if you use a colon to separate the two.
	The hosts are tried in the order listed until a successful connection is made.
	Note: A suitable representation for including a literal IPv6[10] address in the host name parameter is desired, but has not yet been determined or implemented in practice.
portno	Contains the TCP port number of the directory you would like to connect to. The default LDAP port of 3060 can be obtained by supplying the constant LDAP_PORT. If a host includes a port number, this parameter is ignored.

`ldap_init()` and `ldap_open()` both return a session handle, or pointer, to an opaque structure that must be passed to subsequent calls to the session. These routines return `NULL` if the session cannot be initialized. You can check the error reporting mechanism for your operating system to determine why the call failed.

2.4.2 Initializing the Session by Using DBMS_LDAP

In the PL/SQL API, the function `DBMS_LDAP.init()` initiates an LDAP session.

The syntax of the function is as follows:

```
FUNCTION init (hostname IN VARCHAR2, portnum  IN PLS_INTEGER )

RETURN SESSION;
```

The function `init` requires a valid host name and port number to establish an LDAP session. It allocates a data structure for this purpose and returns a handle of the type `DBMS_LDAP.SESSION` to the caller. The handle returned from the call should be used in all subsequent LDAP operations defined by DBMS_LDAP for the session. The API uses these session handles to maintain state about open connections, outstanding requests, and other information.

A single database session can obtain as many LDAP sessions as required, although the number of simultaneous active connections is limited to 64. One database session typically has multiple LDAP sessions when data must be obtained from multiple servers simultaneously or when open sessions that use multiple LDAP identities are required.

> **Note:**
>
> The handles returned from calls to `DBMS_LDAP.init()` are dynamic constructs. They do not persist across multiple database sessions. Attempting to store their values in a persistent form, and to reuse stored values at a later stage, can yield unpredictable results.

2.4.3 Initializing the Session by Using JNDI

The `oracle.ldap.util.jndi` package supports basic connections by providing wrapper methods for the JNDI implementation.

If you want to use the JNDI to establish a connection, see the following link:

http://www.oracle.com/technetwork/java/jndi/index.html

Here is an implementation of `oracle.ldap.util.jndi` that establishes a non-SSL connection:

```
import oracle.ldap.util.jndi
import javax.naming.*;

public static void main(String args[])
{
  try{
      InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                    args[1],  // port
                                                    args[2],  // DN
                                                    args[3];  // password)

      // Do work
    }
  catch(NamingException ne)
  {
    // javax.naming.NamingException is thrown when an error occurs
  }
}
```

> **Note:**
>
> • `DN` and `password` represent the bind DN and password. For anonymous binds, set these to `""`.
>
> • You can use `ConnectionUtil.getSSLDirCtx()` to establish a no-authentication SSL connection.

2.5 Authenticating an LDAP Session

Individuals or applications seeking to perform operations against an LDAP server must first be authenticated. If the `dn` and `passwd` parameters of these entities are null, the LDAP server assigns a special identity, called anonymous, to these users. Typically, the anonymous user is the least privileged user of the directory.

When a bind operation is complete, the directory server remembers the new identity until another bind occurs or the LDAP session terminates (`unbind_s`). The LDAP server uses the identity to enforce the security model specified by the enterprise in which it is deployed. The identity helps the LDAP server determine whether the user or application identified has sufficient privileges to perform search, update, or compare operations in the directory.

Note that the password for the bind operation is sent over the network in clear text. If your network is not secure, consider using SSL for authentication and other LDAP operations that involve data transfer.

This section contains the following topics:

- Authenticating an LDAP Session Using C API
- Authenticating an LDAP Session Using DBMS_LDAP

2.5.1 Authenticating an LDAP Session Using C API

The C function `ldap_simple_bind_s()` enables users and applications to authenticate to the directory server using a DN and password.

The following code of `ldap_simple_bind_s()` function allows you to authenticate:

```
int ldap_simple_bind_s
(
LDAP* ld,
char* dn,
char* passwd
);
```

Table 2-5 lists and describes the parameters for this function.

Table 2-5   Arguments for ldap_simple_bind_s()

Argument	Description
`ld`	A valid LDAP session handle
`dn`	The identity that the application uses for authentication
`passwd`	The password for the authentication identity

If the `dn` and `passwd` parameters for are `NULL`, the LDAP server assigns a special identity, called anonymous, to the user or application.

2.5.2 Authenticating an LDAP Session Using DBMS_LDAP

The PL/SQL function `simple_bind_s` enables users and applications to use a DN and password to authenticate to the directory.

`simple_bind_s` has this syntax:

```
FUNCTION simple_bind_s ( d IN SESSION, dn IN VARCHAR2, passwd IN VARCHAR2)

RETURN PLS_INTEGER;
```

Note that this function requires as its first parameter the LDAP session handle obtained from `init`.

The following PL/SQL code snippet shows how the PL/SQL initialization and authentication functions just described might be implemented.

```
DECLARE

retval     PLS_INTEGER;
my_session     DBMS_LDAP.session;

BEGIN

retval     := -1;
-- Initialize the LDAP session
my_session     := DBMS_LDAP.init('yow.example.com',3060);
--Authenticate to the directory
retval     :=DBMS_LDAP.simple_bind_s(my_session,'cn=orcladmin',     'welcome');
```

In the previous example, an LDAP session is initialized on the LDAP server `yow.example.com`. This server listens for requests at TCP/IP port number `3060`. The identity `cn=orcladmin`, whose password is `welcome`, is then authenticated. After authentication is complete, regular LDAP operations can begin.

2.6 Searching the Directory

Searches are the most common LDAP operations. Applications can use complex search criteria to select and retrieve entries from the directory.

Search filters and types are described in the following topics:

- Steps Involved in Search-Related Operations
- Options for Directory Server Search Scope
- Filters for Directory Server Search
- Searching the Directory by Using the C API
- Searching the Directory by Using DBMS_LDAP

> **✏ Note:**
>
> This release of the `DBMS_LDAP` API provides only synchronous search capability. This means that the caller of the search functions is blocked until the LDAP server returns the entire result set.

2.6.1 Steps Involved in Search-Related Operations

A typical search operation results in exact information when given proper attributes.

The programming required to initiate a typical search operation and retrieve results can be broken down into the following steps:

1. Decide what attributes must be returned; then place them into an array.

2. Initiate the search, using the scope options and filters of your choice.

3. Obtain an entry from result set.

4. Obtain an attribute from the entry obtained in step 3.

5. Obtain the values of the attributes obtained in step 4; then copy these values into local variables.

6. Repeat step 4 until all attributes of the entry are examined.

7. Repeat Step 3 until there are no more entries

Figure 2-4 uses a flowchart to represent these steps.

Figure 2-4   Flow of Search-Related Operations
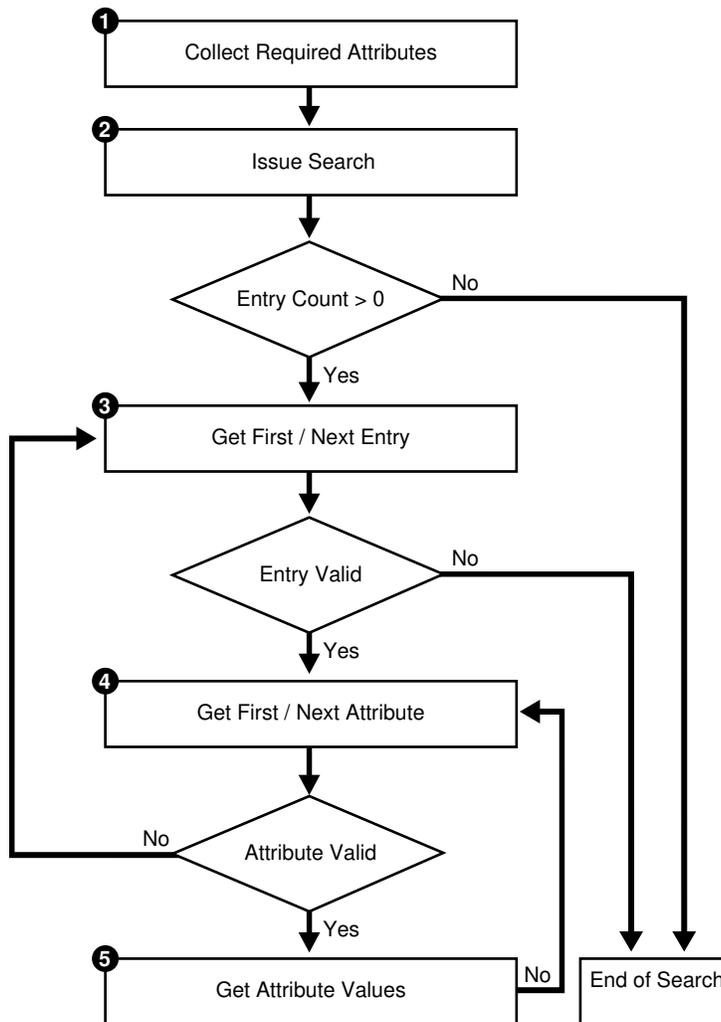
2.6.2 Options for Directory Server Search Scope

The scope of a search determines how many entries the directory server examines relative to the search base.

You can choose one of the three options described in Table 2-6 and illustrated in Figure 2-5.

Table 2-6   Options for search_s() or search_st() Functions

Option	Description
SCOPE_BASE	The directory server looks only for the entry corresponding to the search base.
SCOPE_ONELEVEL	The directory server confines its search to the entries that are the immediate children of the search base entry.

Table 2-6   (Cont.) Options for search_s() or search_st() Functions

Option	Description
SCOPE_SUBTREE	The directory server looks at the search base entry and the entire subtree beneath it.

Figure 2-5   The Three Scope Options
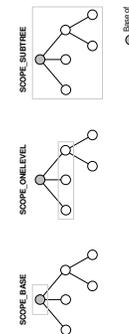
In Figure 2-5, the search base is the shaded circle. The shaded rectangle identifies the entries that are searched.

2.6.3 Filters for Directory Server Search

A search filter is an expression that enables you to confine your search to certain types of entries. The search filter required by the search_s() and search_st() functions follows the string format defined in RFC 1960 of the Internet Engineering Task Force (IETF).

As Table 2-7 shows, there are six kinds of search filters. These are entered in the format *attribute operator value*.

Table 2-7   Search Filters

Filter Type	Format	Example	Matches
Equality	(*att=value*)	(sn=Keaton)	Surnames exactly equal to Keaton.
Approximate	(*att~=value*)	(sn~=Ketan)	Surnames approximately equal to Ketan.
Substring	(attr=[*leading*]*[*any*]*[*trailing*]	(sn=*keaton*)	Surnames containing the string keaton.
		(sn=keaton*)	Surnames starting with keaton.
		(sn=*keaton)	Surnames ending with keaton.
		(sn=ke*at*on)	Surnames starting with ke, containing at and ending with on.

Table 2-7   (Cont.) Search Filters

Filter Type	Format	Example	Matches
Greater than or equal	`attr>=value`	`(sn>=Keaton)`	Surnames lexicographically greater than or equal to `Keaton`.
Less than or equal	`(attr<=value)`	`(sn<=Keaton)`	Surnames lexicographically less than or equal to `Keaton`.
Presence	`(attr=*)`	`(sn=*)`	All entries having the `sn` attribute.
Extensible	`attr[:dn]:=value`	`(sn:dn:=Mary Smith)`	All entries with a surname attribute in the entry or in the DN of the entry matching "Mary Smith".

> **Note:**
>
> - While Oracle Internet Directory supports extensible filters, `ldapsearch` and the Oracle LDAP API do not. You must use a different API, such as JNDI, to use this type of filter.
>
> - Oracle Internet Directory does not support extensible matching using matching rules specified in the filter.

You can use boolean operators and prefix notation to combine these filters to form more complex filters. Table 2-8 provides examples. In these examples, the `&` character represents AND, the `|` character represents OR, and the `!` character represents NOT.

Table 2-8   Boolean Operators

Filter Type	Format	Example	Matches
AND	`(&(filter1)(filter2)). . .)`	`(&(sn=keaton)(objectclass=inetOrgPerson))`	Entries with surname of `Keaton` and object class of `InetOrgPerson`.
OR	`(|(filter1)(filter2)). . .)`	`(|(sn~=ketan)(cn=*keaton))`	Entries with surname approximately equal to `ketan` or common name ending in `keaton`.
NOT	`(!(filter))`	`(!(mail=*))`	Entries without a mail attribute.

The complex filters in Table 2-8 can themselves be combined to create even more complex, nested filters.

> **✎ See Also:**
>
> - The LDAP Filter Definition appendix in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for more information about LDAP filters.
> - RFC 2254 at: http://www.ietf.org

2.6.4 Searching the Directory by Using the C API

The C function `ldap_search_s()` performs a synchronous search of the directory.

You use the following syntax of `ldap_search_s()` function to perform a search:

```
int ldap_search_s
(
LDAP*        ld,
char*        base,
int          scope,
char*        filter
int          attrsonly,
LDAPMessage** res
);
```

Table 2-9 describes the arguments in details.

Table 2-9   Arguments for ldap_search_s()

Argument	Description
ld	A valid LDAP session handle
base	The DN of the search base.
scope	The breadth and depth of the DIT to be searched.
filter	The filter used to select entries of interest.
attrs	The attributes of interest in the entries returned.
attrso	If set to 1, only returns attributes.
res	This argument returns the search results.

`ldap_search_s` works with several supporting functions to refine the search. The steps that follow show how all of these C functions fit into the program flow of a search operation. C API Reference , examines all of these functions in depth.

1. Decide what attributes must be returned; then place them into an array of strings. The array must be null terminated.

2. Initiate the search, using `ldap_search_s()`. Refine your search with scope options and filters.

3. Obtain an entry from the result set, using either the `ldap_first_entry()` function or the `ldap_next_entry()` function.

4. Obtain an attribute from the entry obtained in step 3. Use either the `ldap_first_attribute()` function or the `ldap_next_attribute()` function for this purpose.

5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use the `ldap_get_values()` function or the `ldap_get_values_len()` function for this purpose.

6. Repeat step 4 until all attributes of the entry are examined.

7. Repeat step 3 until there are no more entries.

2.6.5 Searching the Directory by Using DBMS_LDAP

You use the function `DBMS_LDAP.search_s()` to perform directory searches if you use the PL/SQL API.

Here is the syntax for `DBMS_LDAP.search_s()`:

```
FUNCTION search_s

(
ld       IN  SESSION,
base     IN  VARCHAR2,
scope    IN  PLS_INTEGER,
filter   IN  VARCHAR2,
attrs    IN  STRING_COLLECTION,
attronly IN  PLS_INTEGER,
res      OUT MESSAGE
)

RETURN PLS_INTEGER;
```

The function takes the arguments listed and described in Table 2-10.

Table 2-10   Arguments for DBMS_LDAP.search_s() and DBMS_LDAP.search_st()

Argument	Description
`ld`	A valid session handle
`base`	The DN of the base entry in the LDAP server where search should start
`scope`	The breadth and depth of the DIT that must be searched
`filter`	The filter used to select entries of interest
`attrs`	The attributes of interest in the entries returned

Table 2-10   (Cont.) Arguments for DBMS_LDAP.search_s() and DBMS_LDAP.search_st()

Argument	Description
`attronly`	If set to `1`, only returns the attributes
`res`	An `OUT` parameter that returns the result set for further processing

`search_s` works with several supporting functions to refine the search. The steps that follow show how all of these PL/SQL functions fit into the program flow of a search operation.

1. Decide what attributes need to be returned; then place them into the `DBMS_LDAP.STRING_COLLECTION` data-type.

2. Perform the search, using either `DBMS_LDAP.search_s()` or `DBMS_LDAP.search_st()`. Refine your search with scope options and filters.

3. Obtain an entry from the result set, using either `DBMS_LDAP.first_entry()` or `DBMS_LDAP.next_entry()`.

4. Obtain an attribute from the entry obtained in step 3. Use either `DBMS_LDAP.first_attribute()` or `DBMS_LDAP.next_attribute()` for this purpose.

5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use either `DBMS_LDAP.get_values()` or `DBMS_LDAP.get_values_len()` for this purpose.

6. Repeat step 4 until all attributes of the entry are examined.

7. Repeat step 3 until there are no more entries.

2.7 Terminating the Session

You can terminate a session by C API or DBMS_LDAP API.

- C API to Terminate a Session
- DBMS_LDAP API to Terminate a Session

2.7.1 C API to Terminate a Session

After an LDAP session handle is obtained and all directory-related work is complete, the LDAP session must be destroyed. In the C API, the `ldap_unbind_s()` function is used for this purpose.

`ldap_unbind_s()` has this syntax:

```
int ldap_unbind_s
(
    LDAP*     ld
);
```

A successful call to `ldap_unbind_s()` closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer

`LDAP_SUCCESS` to its callers. After `ldap_unbind_s()` is invoked, no other LDAP operations are possible. A new session must be started with `ldap_init()`.

2.7.2 DBMS_LDAP API to Terminate a Session

The `DBMS_LDAP.unbind_s()` function destroys an LDAP session if the PL/SQL API is used.

`unbind_s` has the following syntax:

```
FUNCTION unbind_s (ld IN SESSION )  RETURN PLS_INTEGER;
```

`unbind_s` closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer `DBMS_LDAP.SUCCESS` to its callers. After the `unbind_s` is invoked, no other LDAP operations are possible. A new session must be initiated with the `init` function.

3

Extensions to the LDAP Protocol

There are extensions to LDAP protocol available in Oracle Internet Directory starting 11g Release 1 (11.1.1) which helps in authentication mechanism, controls, using proxy etc. The following topics describe extensions to LDAP protocol in detail:

- Using SASL Authentication Mechanism
- Understanding Oracle Internet Directory Controls
- Using Proxy on Behalf of End Users
- Creating Dynamic Password Verifiers
- Performing Hierarchical Searches Using CONNECT_BY Control
- Understanding Sorted LDAP Search Results
- Understanding Paged LDAP Search Results
- Using Password Policies

3.1 Using SASL Authentication Mechanism

Oracle Internet Directory supports SASL DIGEST-MD5 Authentication and SASL external authentication.

The following sections describe the two methods:

- Working with SASL DIGEST-MD5 Authentication Mechanism
- Understanding SASL External Authentication Mechanism

3.1.1 Working with SASL DIGEST-MD5 Authentication Mechanism

SASL Digest-MD5 authentication is the required authentication mechanism for LDAP Version 3 servers (RFC 2829). LDAP Version 2 does not support Digest-MD5.

To use the Digest-MD5 authentication mechanism, you can use either the Java API or the C API to set up the authentication. The C API supports only `auth` mode.

> ✏️ **See Also:**
>
> - Java-specific information in Using DIGEST-MD5 to Perform SASL Authentication and Using SASL Digest-MD5 auth-int and auth-conf Modes.
> - C-specific information in "Authenticating to the Directory" and SASL Authentication Using Oracle Extensions

This section contains the following topics:

- Modes of SASL Digest-MD5 Mechanism

- Using SASL Digest-MD5 Mechanism

- Steps Involved in SASL DIGEST-MD5 Authentication Mechanism

3.1.1.1 Modes of SASL Digest-MD5 Mechanism

The SASL Digest-MD5 mechanism includes three modes, each representing a different security level or "Quality of Protection."

The following table describes modes of SASL Digest-MD5:

Table 3-1   Modes of SASL Digest-MD5

Modes	Description
`auth`	Authentication only. Authentication is required only for the initial bind. After that, information is passed in clear text.
`auth-int`	Authentication plus integrity. Authentication is required for the initial bind. After that, check sums are used to guarantee the integrity of the data.
`auth-conf`	Authentication plus confidentiality. Authentication is required for the initial bind. After that, encryption is used to protect the data. Five cipher choices are available: • DES • 3DES • RC4 • RC4-56 • RC-40 These are all symmetric encryption algorithms.

Prior to 10*g* (10.1.4.0.1), Oracle Internet Directory supported only the `auth` mode of the Digest-MD5 mechanism. As of 10*g* (10.1.4.0.1), Oracle Internet Directory supports all three modes with the Java Naming and Directory Interface (JNDI) of jdk1.4 API or with the OpenLDAP Java API. The Oracle LDAP SDK supports only `auth` mode.

3.1.1.2 Using SASL Digest-MD5 Mechanism

Out of the box, Oracle Internet Directory SASL Digest-MD5 authentication supports generation of static SASL Digest-MD5 verifiers based on user or password, but not based on realm. If you want to use SASL Digest-MD5 with realms, you must enable reversible password generation by changing the value of the `orclpasswordencryptionenable` attribute to 1 in the related password policy before provisioning new users. The LDIF file for modifying the value should look like this:

```
dn: cn=default,cn=pwdPolicies,cn=Common,cn=Products,cn=OracleContext
changetype: modify
replace: orclpwdencryptionenable
orclpwdencryptionenable: 1
```

The Digest-MD5 mechanism is described in RFC 2831 of the Internet Engineering Task Force. It is based on the HTTP Digest Authentication (RFC 2617).

> **See Also:**
>
> - Internet Engineering Task Force Web site, at `http://www.ietf.org`.
> - Open LDAP class libraries `http://www.openldap.org`.

3.1.1.3 Steps Involved in SASL DIGEST-MD5 Authentication Mechanism

SASL Digest-MD5 authentication involves many steps and finally the directory server verifies the client's response.

The following steps describe SASL Digest-MD5 authentication in detail:

1. The directory server sends data that includes various authentication options that it supports and a special token to the LDAP client.

2. The client responds by sending an encrypted response that indicates the authentication options that it has selected. The response is encrypted in such a way that proves that the client knows its password.

3. The directory server then decrypts and verifies the client's response.

3.1.2 Understanding SASL External Authentication Mechanism

Oracle Internet Directory provides the SASL external mechanism over an SSL mutual connection. The authorization identity (DN) is derived from the client certificate during the SSL network negotiation.

The following is from section 7.4 of RFC 2222 of the Internet Engineering Task Force.

The mechanism name associated with external authentication is "EXTERNAL". The client sends an initial response with the authorization identity. The server uses information, external to SASL, to determine whether the client is authorized to authenticate as the authorization identity. If the client is so authorized, the server indicates successful completion of the authentication exchange; otherwise the server indicates failure.

The system providing this external information may be, for example, IPsec or SSL/TLS.

If the client sends the empty string as the authorization identity (thus requesting the authorization identity be derived from the client's authentication credentials), the authorization identity is to be derived from authentication credentials that exist in the system which is providing the external authentication.

3.2 Understanding Oracle Internet Directory Controls

The LDAPv3 Protocol, as defined by RFC 2251, allows extensions by means of controls.Oracle Internet Directory supports several controls. Some are standard and described by RFCs.

Other controls, such as the CONNECT_BY control for hierarchical searches are Oracle-specific. You can use controls with either Java or C.

This section contains the following topics:

- Overview of Oracle Internet Directory Controls

- Controls Supported by Oracle Internet Directory
- Viewing Supported Oracle Internet Directory Controls

3.2.1 Overview of Oracle Internet Directory Controls

Controls can be sent to a server or returned to the client with any LDAP message. These controls are referred to as server controls. The LDAP API also supports a client-side extension mechanism through the use of client controls. These controls affect the behavior of the LDAP API only and are never sent to a server.

For information about using LDAP controls in C, see Working With Controls

For information about using LDAP controls in Java, see the documentation for the JNDI package `javax.naming.ldap` at: `http://www.oracle.com/technetwork/java/jndi/index.html`

3.2.2 Controls Supported by Oracle Internet Directory

Oracle Internet Directory supports Request and Response controls.

- Request Controls Supported by Oracle Internet Directory
- Response Controls Supported by Oracle Internet Directory

3.2.2.1 Request Controls Supported by Oracle Internet Directory

Table 3-2 lists the request controls supported by Oracle Internet Directory

Table 3-2   Request Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
2.16.840.1.113730.3.4.1 8	Proxy Authorization	Allows an LDAP client application to bind to Oracle Internet Directory server with its own identity and then to perform operations on behalf of another user or on behalf of multiple users.
		This control can improve performance especially for proxy operations performed on behalf of multiple users. The LDAP operation does not require a rebind for each user.
		For example, consider this scenario:
		• A client application named `proxy-app` binds to Oracle Internet Directory server.
		• A user named John Doe sends a request to the `proxy-app` client application. The client must be a member of the proxy authorization group.
		• Instead of passing the request to the Oracle Internet Directory server as `proxy-app`, the client passes the Proxy Authorization control along with the proxy DN (John Doe) to the server.
		• The server performs the LDAP request for John Doe using John Doe's privileges (ACLs). The server can access only the information or perform only the operations that John Doe has privileges to access or perform.
		Considerations for the Proxy Authorization control are:
		• If the control is not accompanied by a proxy `dn`, the server throws an exception and returns an error to the client. If the criticality flag is not set to `TRUE`, the request is rejected with a protocol error.
		• If the proxy user has the privileges to perform the operation as per the ACLs, the server executes the request; otherwise, the server returns the result code "123".
		• Oracle Internet Directory server determines if a Proxy Authorization request is to be honored. Anonymous users are not allowed to assume the identity of others.
		• If the server receives more than one Proxy Authorization control in the same operation, an error is returned to the client application and the operation attempt is unsuccessful.

Table 3-2   (Cont.) Request Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
1.3.6.1.4.1.42.2.27.8.5.1	Password Policy	Allows an LDAP client to request information from Oracle Internet Directory server about the current password policy state for a user. If a password policy is applicable, a client can send this control with these operations: • `ldapbind`, `ldapmodify`, `ldapadd`, or `ldapcompare` • `ldapsearch` Other operations such as `ldapunbind` and `ldapabandon` are not supported. Also, this control does not apply to a replication bind. The password policy request control does not have a `controlValue`, but the response control has return values, which are described in Table 3-3. The password policy is typically applied to the single-valued attribute `userPassword` for the LDAP simple authentication method or password-based Simple Authentication and Security Layer (SASL) authentication. For more information, see "Password Policy for LDAP Directories" at this location: http://tools.ietf.org/html/draft-behera-ldap-password-policy-09

2.16.840.1.113730.3.4.3	Persistent Search	Allows an LDAP client to send a persistent search request to Oracle Internet Directory server.
		
		A persistent search operation is an enhanced search that continues after the initial search results are returned by the server to the client. After the initial search is finished, the connection to the server is kept alive until the client unbinds or abandons the operation. The client can track changes for entries in the search scope and receive an Entry Change Notification response control if an entry is modified.
		
		The definition for this control is:
		
		`PersistentSearch ::= SEQUENCE {`
		`    controlType 2.16.840.1.113730.3.4.3`
		`    changeTypes INTEGER,`
		`    changesOnly BOOLEAN,`
		`    returnECs BOOLEAN`
		`}`
		
		For a description of these fields, see:
		
		http://tools.ietf.org/id/draft-ietf-ldapext-psearch-03.txt

Table 3-2   (Cont.) Request Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
2.16.840.1.113894.1.8.39	Computed Attribute Value Uniqueness	Allows computed attribute value uniqueness for an entry-based combination of multiple attribute values. Usually, attribute uniqueness is configured only for a single attribute. An application that requires uniqueness for a combination of attribute values can send this control, and Oracle Internet Directory server ensures that the computed attribute value is unique across the directory during an operation such as `ldapadd`, `ldapmodify -add`, or `ldapmodify -replace`. If an entry with the computed value already exists, Oracle Internet Directory server returns the LDAP error code: LDAP_ALREADY_EXISTS = 0x44. For example, in a multi-tenant environment, the UID attribute of a user must be unique for a specific tenant but not necessarily unique across the directory. An application creates an LDAP entry with a computed attribute named TenantID_UID, where TenantID is the identifier of the tenant and UID is the attribute. The application that creates the LDAP entry sends this control, and if an entry with the computed attribute already value already exists, Oracle Internet Directory server returns the LDAP_ALREADY_EXISTS error code.
2.16.840.1.113730.3.4.9	OID_SEARCH_VLV_REQ_CONTROL	Allows a client to specify that the server return, for a given LDAP search, a contiguous subset of a large search result set. It can be used to go through the search results one page at a time, which allows a client to retrieve results more quickly and prevents the client from needing to store too many search results at a time. The server returns the OID_SEARCH_VLV_RES_CONTROL 2.16.840.1.113730.3.4.10. The OID_SEARCH_VLV_REQ_CONTROL works only in conjunction with the SORT control (ldapsearch -T argument). If you do not include a SORT control, the request returns LDAP error code 53 - Search operation with VLV request control is missing SORT request control. The SORT control can contain any sort specification valid for the server. When the SORT control is used with the OID_SEARCH_VLV_REQ_CONTROL, the server does not return the complete set of sorted search results, but instead returns a contiguous subset of those entries specified in the control using a target entry as a reference point for results. For more information, see "LDAP Extensions for Scrolling View Browsing of Search Results" at `http://tools.ietf.org/html/draft-ietf-ldapext-ldapv3-vlv-09`.
1.2.840.113556.1.4.319	OID_SEARCH_PAGING_CONTROL	See "Understanding Paged LDAP Search Results"
1.2.840.113556.1.4.473	OID_SEARCH_SORTING_REQUEST_CONTROL	See "Understanding Sorted LDAP Search Results"

Table 3-2   (Cont.) Request Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
2.16.840.1.113730.3.4.2	GSL_MANAGE_DSA_CONTROL	Used to manage referrals, dynamic groups, and alias objects in Oracle Internet Directory. For more information, please see RFC 3296, "Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories," at http://www.ietf.org.
2.16.840.1.113894.1.8.1	OID_RESET_PROXYCONTROL_IDENTITY	Used to perform a proxy switch of an identity on an established LDAP connection. For example, suppose that Application A connects to the directory server and then wishes to switch to Application B. It can simply do a rebind by supplying the credentials of Application B. However, there are times when the proxy mechanism for the application to switch identities could be used even when the credentials are not available. With this control, Application A can switch to Application B provided Application A has the privilege in Oracle Internet Directory to proxy as Application B.
2.16.840.1.113894.1.8.2	OID_APPLYUSEPASSWORD_POLICY	Sent by applications that require Oracle Internet Directory to check for account lockout before sending the verifiers of the user to the application. If Oracle Internet Directory detects this control in the verifier search request and the user account is locked, then Oracle Internet Directory does not send the verifiers to the application. It sends an appropriate password policy error.
2.16.840.1.113894.1.8.3	CONNECT_BY	See Performing Hierarchical Searches Using CONNECT_BY Control.
2.16.840.1.113894.1.8.4	OID_CLIENT_IP_ADDRESS	Intended for a client to send the end user IP address if IP lockout is to be enforced by Oracle Internet Directory.
2.16.840.1.113894.1.8.5	GSL_REQDATTR_CONTROL	Used with dynamic groups. Directs the directory server to read the specific attributes of the members rather than the membership lists.
2.16.840.1.113894.1.8.6	PasswordStatusRequestControl	When packaged as part of the LDAP Bind/Compare operation request, this control causes the server to generate a password policy response control. The actual response control depends on the situation. Cases include imminent password expiration, number of grace logins remaining, password expired, and account locked.
2.16.840.1.113894.1.8.14	OID_DYNAMIC_VERIFIER_REQUEST_CONTROL	The request control that the client sends when it wants the server to create a dynamic password verifier. The server uses the parameters in the request control to construct the verifier.
2.16.840.1.113894.1.8.16	AccountStatusRequestControl	When packaged with the LDAP search operation associated with the authentication process, the Oracle Internet Directory returns a password policy response control to inform the client application of account state related information like account lockout, password expiration etc. The application can then parse and enforce the results.
2.16.840.1.113894.1.8.23	GSL_CERTIFICATE_CONTROL	Certificate search control. The request control that the client sends to specify how to search for a user certificate. See the appendix Searching the Directory for User Certificates in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*.

Table 3-2   (Cont.) Request Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
2.16.840.1.113894.1.8.29	EffectivePolicyControl	This control is packaged as part of an LDAP base search, where the base DN is that of the user entry being tested. The entry need not exist in the directory at the time. Passing this control results in the return of the LDAP entry describing the applicable password policy, assuming the entity performing the search has the access rights to view the password policy entry. If the desired password is provided as the optional testPassword parameter, the directory server returns the response control 2.16.840.1.113894.1.8.32.
2.16.840.1.113894.1.8.36	DelSubtreeControl	When this control is sent with a delete operation, it causes the deletion of the entire subtree below the DN provided. Any user having necessary privileges can perform this operation.
1.3.6.1.1.21.2	Transaction Specification Control	This is an LDAPControl indicating association of an operation to a transaction by means of the transaction identifier, which is the value of this control. Its criticality is TRUE.

3.2.2.2 Response Controls Supported by Oracle Internet Directory

Table 3-3 lists the response controls supported by Oracle Internet Directory

Table 3-3   Response Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
1.3.6.1.4.1.42.2.27.8.5.1	Password Policy	Response control that the Oracle Internet Directory server returns to an LDAP client in response to the Password Policy request control. The response control value is encoded as follows:

```
PasswordPolicyResponseValue ::= SEQUENCE {
  warning [0] CHOICE {
  timeBeforeExpiration [0] INTEGER (0 .. maxInt),
  graceAuthNsRemaining [1] INTEGER (0 .. maxInt) }
OPTIONAL,
  error   [1] ENUMERATED {
  passwordExpired            (0),
  accountLocked              (1),
  changeAfterReset           (2),
  passwordModNotAllowed      (3),
  mustSupplyOldPassword      (4), /*currently not
supported*/
  insufficientPasswordQuality (5),
  passwordTooShort           (6),
  passwordTooYoung           (7),
  passwordInHistory          (8) } OPTIONAL }
```

The Password Policy response control does not offer the `mustSupplyOldPassword` error code, because Oracle Internet Directory does not support the `pwdsafemodify` functionality.

The server sends either an error or warning in the password policy response control (but not both). Error codes are described in:

http://tools.ietf.org/html/draft-behera-ldap-password-policy-09

Control criticality is not returned in the response. For a non violation of the password policy, the server does not send a response.

Table 3-3   (Cont.) Response Controls Supported by Oracle Internet Directory

Object Identifier	Name	Description
| 2.16.840.1.113730.3.4 .7 | Entry Change Notification | Returned by the Oracle Internet Directory server to an LDAP client in response to a Persistent Search control that has the `returnECs` field set to TRUE. This control is returned for each changed entry that matches the persistent search criteria and describes the change made to the specific search entry.<br><br>The definition for this control is:<br><br>```
EntryChangeNotification ::= SEQUENCE {
    controlType 2.16.840.1.113730.3.4.7
    changeType ENUMERATED {
        add             (1),
        delete          (2),
        modify          (4),
        modDN           (8)
    },
    previousDN    LDAPDN OPTIONAL,      -- modifyDN
ops. only
        changeNumber INTEGER OPTIONAL     -- if supported
}
```<br><br>For a description of these fields, see:<br>http://tools.ietf.org/id/draft-ietf-ldapext-psearch-03.txt |
2.16.840.1.113730.3.4 .10	OID_SEARCH_VLV_RE S_CONTROL	The server sends this control in response to an OID_SEARCH_VLV_REQ_CONTROL 2.16.840.1.113730.3.4.9.
2.16.840.1.113894.1.8 .7	OID_PASSWORD_EXP WARNING_CONTROL	Password policy control. Response control that the server sends when the pwdExpireWarning attribute is enabled and the client sends the request control. The response control value contains the time in seconds to password expiration.
2.16.840.1.113894.1.8 .8	OID_PASSWORD_GRA CELOGIN_CONTROL	Password policy control. The response control that the server sends when grace logins are configured and the client sends a request control. The response control value contains the remaining number of grace logins.
2.16.840.1.113894.1.8 .20	OID_PWDEXPIRED_C ONTROL	Password policy control. The response control that the server sends when the password has expired, there are no grace logins remaining, and the client sends a request control.
2.16.840.1.113894.1.8 .9	OID_PASSWORD_MUS TCHANGE_CONTROL	Password policy control. The response control that the server sends when forced password reset is enabled and the client sends the request control. The client must force the user to change the password upon receipt of this control.
2.16.840.1.113894.1.8 .15	OID_DYNAMIC_VERIFI ER_RESPONSE_CON TROL	The response control that the server sends to the client when an error occurs. The response control contains the error code.
2.16.840.1.113894.1.8 .32	PasswordValidationCont rol	The server sends this in response to control 2.16.840.1.113894.1.8.29 when the desired password is provided as the optional testPassword parameter. A client application can parse the validationResult to determine whether the password can be accepted by the server ("Success") or the reason it has been rejected. The same type of error message generated during a failed LDAP modify operation on userpassword is returned as the value.

3.2.3 Viewing Supported Oracle Internet Directory Controls

To find out what controls are available in your Oracle Internet Directory installation, you need to run `ldapsearch` command.

Run the following command to view supported Oracle Directory Controls:

```
ldapsearch -p port -b "" -s base "objectclass=*"
```

Look for entries that begin with `supportedcontrol=`.

3.3 Using Proxy on Behalf of End Users

Often applications must perform operations that require impersonating an end user. An application may, for example, want to retrieve resource access descriptors for an end user.

Resource access descriptors are discussed in The Access Control Directive Format chapter of*Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*.

A proxy switch occurs at run time on the JNDI context. An LDAP v3 feature, proxying can only be performed using `InitialLdapContext`, a subclass of `InitialDirContext`. If you use the Oracle extension `oracle.ldap.util.jndi.ConnectionUtil` to establish a connection (the example following), `InitialLdapContext` is always returned. If you use JNDI to establish the connection, make sure that it returns `InitialLdapContext`.

> **✎ Note:**
>
> To perform the proxy switch to an end user, you need the user's DN.
>
> If there is a User object already available fetched from Subscriber, then the user's DN is available as shown in this fragment:
>
> ```
> //User user = subscriber.getUser(...);
> user.getDn();
> ```
>
> If you do not have the User object, you can create it if you have the user's name, UID, SAM account name, or Kerberos principal are available, as shown in the following fragment:
>
> ```
> User user = new User(dirCtx, idType, userIdValue, subscriber, true);
> // idType can be Util.IDTYPE_SIMPLE / Util.IDTYPE_GUID /
> Util.IDTYPE_WINDOWS / Util.IDTYPE_KERB_PRINCIPALuser.getDn();
> ```

This code shows how the proxy switch occurs:

```
import oracle.ldap.util.jndi.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import javax.naming.*;

public static void main(String args[])
{
```

```
    try{
        InitialLdapContext appCtx=ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                                  args[1], // port
                                                                  args[2], // DN
                                                                  args[3]; // pass)
            // Do work as application
            // . . .
            String userDN=null;
            // assuming userDN has the end user DN value
            // Now switch to end user
            ctx.addToEnvironment(Context.SECURITY_PRINCIPAL, userDN);
            ctx.addToEnvironment("java.naming.security.credentials", "");
            Control ctls[] = {
              new ProxyControl()
            };
            ((LdapContext)ctx).reconnect(ctls);
            // Do work on behalf of end user
            // . . .
        }
    catch(NamingException ne)
        {
         // javax.naming.NamingException is thrown when an error occurs
        }
}
```

The `ProxyControl` class in the code immediately preceding implements a `javax.naming.ldap.Control`. To learn more about LDAP controls, see the LDAP control section of *Oracle Fusion Middleware Reference for Oracle Identity Management*. Here is an example of what the `ProxyControl` class might look like:

```
import javax.naming.*;
import javax.naming.ldap.Control;
import java.lang.*;

public class ProxyControl implements Control {

    public byte[] getEncodedValue() {
        return null;
    }

    public String getID() {
        return "2.16.840.1.113894.1.8.1";
    }

    public boolean isCritical() {
        return false;
    }
}
```

3.4 Creating Dynamic Password Verifiers

You can modify the LDAP authentication APIs to generate application passwords dynamically —that is, when users log in to an application. This feature has been designed to meet the needs of applications that provide parameters for password verifiers only at runtime.

This section contains the following topics:

- Using Request Control to Create Dynamic Password Verifiers
- Syntax for DynamicVerifierRequestControl

- Parameters Required by the Hashing Algorithms
- Understanding Request Control for Dynamic Password Verifiers
- Understanding Response Control for Dynamic Password Verifiers
- Obtaining Privileges for the Dynamic Verifier Framework

3.4.1 Using Request Control to Create Dynamic Password Verifiers

Creating a password verifier dynamically involves modifying the LDAP authentication APIs `ldap_search` or `ldap_modify` to include parameters for password verifiers.

An LDAP control called `DynamicVerifierRequestControl` is the mechanism for transmitting these parameters. It takes the place of the password verifier profile used to create password verifiers statically. Nevertheless, dynamic verifiers, like static verifiers, require that the directory attributes `orclrevpwd` (synchronized case) and `orclunsyncrevpwd` (unsynchronized case) be present and that these attributes be populated.

Note that the `orclpwdencryptionenable` attribute of the password policy entry in the user's realm must be set to `1` if `orclrevpwd` is to be generated. If you fail to set this attribute, an exception is thrown when the user tries to authenticate. To generate `orclunsyncrevpwd`, you must add the crypto type 3DES to the entry `cn=defaultSharedPINProfileEntry,cn=common,cn=products,cn=oraclecontext`.

3.4.2 Syntax for DynamicVerifierRequestControl

`DynamicVerifierRequestControl` LDAP control is the mechanism to transmit parameters for password verifiers.

The request control looks like this:

```
DynamicVerifierRequestControl
controlOid:  2.16.840.1.113894.1.8.14
criticality:  FALSE
controlValue: an OCTET STRING whose value is the BER encoding of the following
type:

ControlValue ::= SEQUENCE {

                version [0]
                crypto  [1] CHOICE OPTIONAL {
                   SASL/MD5  [0] LDAPString,
                   SyncML1.0 [1] LDAPString,
                   SyncML1.1 [2] LDAPString,
                   CRAM-MD5  [3] LDAPString },
                username [1] OPTIONAL LDAPString,
                realm    [2] OPTIONAL LDAPString,
                nonce    [3] OPTIONAL LDAPString,
                }
```

Note that the parameters in the control structure must be passed in the order in which they appear. Table 3-4 defines these parameters.

Table 3-4   Parameters in DynamicVerifierRequestControl

Parameter	Description
controlOID	The string that uniquely identifies the control structure.
crypto	The hashing algorithm. Choose one of the four identified in the control structure.
username	The distinguished name (DN) of the user. This value must always be included.
realm	A randomly chosen realm. It may be the identity management realm that the user belongs to. It may even be an application realm. Required only by the SASL/MD5 algorithm.
nonce	An arbitrary, randomly chosen value. Required by SYNCML1.0 and SYNCML1.1.

3.4.3 Parameters Required by the Hashing Algorithms

Hashing algorithms are used to create dynamic password verifiers.

Table 3-5 lists the four hashing algorithms and the parameters that each algorithm uses as building blocks. Note that, although all algorithms use the user name and password parameters, they differ in their use of the realm and nonce parameters.

Table 3-5   Parameters Required by the Hashing Algorithms

Algorithm	Parameters Required
SASL/MD5	username, realm, password
SYNCML1.0	username, password, nonce
SYNCML1.1	username, password, nonce
CRAM-MD5	username, password

3.4.4 Understanding Request Control for Dynamic Password Verifiers

Applications that require password verifiers to be generated dynamically must include DynamicVerifierRequestControl in their authentication APIs.

Either ldap_search or ldap_compare must incorporate the controlOID and the control values as parameters. They must BER-encode the control values as shown in Syntax for DynamicVerifierRequestControl. Then they must send both controlOID and the control values to the directory server. More description about parameters is described in the following sections:

• About Parameters Required If ldap_search Is Used

• About Parameters Required If ldap_compare Is Used

3.4.4.1 About Parameters Required If ldap_search Is Used

If you want the application to authenticate the user, use `ldap_search` to pass the control structure. If `ldap_search` is used, the directory passes the password verifier that it creates to the client.

`ldap_search` must include the DN of the user, the `controlOID`, and the control values. If the user's password is a single sign-on password, the attribute passed is `authpassword`. If the password is a numeric pin or another type of unsynchronized password, the attribute passed is `orclpasswordverifier;orclcommonpin`.

3.4.4.2 About Parameters Required If ldap_compare Is Used

If you want Oracle Internet Directory to authenticate the user, use `ldap_compare` to pass the control structure. In this case, the directory retains the verifier and authenticates the user itself.

Like `ldap_search`, `ldap_compare` must include the DN of the user, the `controlOID`, the control values, and the user's password attribute. For `ldap_compare`, the password attribute is `orclpasswordverifier;orclcommonpin` (unsynchronized case).

3.4.5 Understanding Response Control for Dynamic Password Verifiers

When it encounters an error, the directory sends the LDAP control `DynamicVerifierResponseControl` to the client.This response control contains the error code.

To learn about the error codes that the response control sends, see the Troubleshooting Dynamic Password Verifiers in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*.

3.4.6 Obtaining Privileges for the Dynamic Verifier Framework

If you want the directory to create password verifiers dynamically, you must add your application identity to the VerifierServices group of directory administrators.

If you fail to perform this task, the directory returns an `LDAP_INSUFFICIENT_ACCESS` error.

3.5 Performing Hierarchical Searches Using CONNECT_BY Control

One of the server controls you can pass to an LDAP search function is `CONNECT_BY`. This is an Oracle-specific control that causes the search to traverse a hierarchy.

For example, if you search for all the users in `group1`, without the `CONNECT_BY` control, the search function returns only users who are direct members of `group1`. If you pass the `CONNECT_BY` control, however, the search function traverses the hierarchy. If `group2` is a member of `group1`, the search also returns users in `group2`. If group3 is a member

of `group2`, the search also returns users in `group3`, and so forth. More information on search function is explained in the following sections:

- About Characteristic Features of the CONNECT_BY Control
- Value Fields in the CONNECT_BY Control
- Using CONNECT_BY Control

3.5.1 About Characteristic Features of the CONNECT_BY Control

`CONNECT_BY` control is used to search through all containers in which an entry is contained.

In 10*g* (10.1.4.0.1), the `CONNECT_BY` control was enhanced in two ways:

- You can now traverse the hierarchy in either direction. That is, you can search through all containers in which an entry is contained, and through all containers contained within an entry.
- You can now specify the number of levels of the hierarchy to search.

3.5.2 Value Fields in the CONNECT_BY Control

In previous releases, the `CONNECT_BY` control required no values.

Because of the new functionality, you can now pass one or both of the following values to `CONNECT_BY`:

Table 3-6   Connect_By Control values

Values	Description
Hierarchy-establishing attribute	A string representing the attribute to be searched. This value is necessary only when searching through all containers in which an entry is contained. When searching through containers contained within an entry, you need not provide this value because the search filter provides that information.
Number of levels	An integer representing the number of levels to traverse. If the value is `0`, the search traverses all levels. The default value is `0`, so you need not pass this value if you want the search to traverse all levels.

3.5.3 Using CONNECT_BY Control

You can use CONNECT_BY control to find all the user and group related information.

The following topics describe how to use the CONNECT_BY control:

- Finding All the Groups to Which a User Belongs
- Finding Only the Groups to Which a User Directly Belongs
- Finding All Members of a Group
- Finding all Managers of a User

> **✎ See Also:**
>
> - "ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s"
> - "Working With Controls"

3.5.3.1 Finding All the Groups to Which a User Belongs

Using a filter such as `(member=cn=jsmith)`, you do not need to provide the hierarchy-establishing attribute member because it is in the search filter. You do not need to pass a value for the number of levels because `0` is the default.

3.5.3.2 Finding Only the Groups to Which a User Directly Belongs

Using the same filter as in Example 1, you would pass the integer control value `1`. The result would be the same as if you did not use the `CONNECT_BY` control at all.

3.5.3.3 Finding All Members of a Group

In this case, your search filter would specify `(objectclass=*)`, but if you want to find all members of `group1`, the attribute for traversing the hierarchy is `member`. For this search, you must pass the string `"member"` as the hierarchy-establishing attribute. You do not need to pass a value for the number of levels because `0` is the default.

3.5.3.4 Finding all Managers of a User

This is similar to Example 3, except that you want to find all managers of the user `jsmith`, so `manager` is the attribute for traversing the hierarchy. For this search, you would pass the string `"manager"`. You do not need to pass a value for the number of levels because `0` is the default.

3.6 Understanding Sorted LDAP Search Results

As of Oracle Internet Directory *g* (10.1.4.0.1), you can obtain sorted results from an LDAP search, as described by IETF RFC 2891. You request sorted results by passing a control of type 1.2.840.113556.1.4.473 to the search function. The server returns a response control is of type 1.2.840.113556.1.4.474. Error processing and other details are described in RFC 2891.

> **✎ See Also:**
>
> IETF RFC 2891, "LDAP Control Extension for Server Side Sorting of Search Results," at `http://www.ietf.org`.

Sorting and paging may be used together.

The Oracle Internet Directory implementation of RFC 2891 has the following limitations:

- It supports only one `attributeType` in the control value.

- It uses the default ordering rule defined in the schema for each attribute.

- Linguistic sorting is not supported.

- The default sorting order is ascending.

- If a sort key is a multi-valued attribute, and an entry has multiple values for that attribute, and there are no other controls that affect the sorting order, then the server uses the least value, according to the ordering rule for that attribute.

- The sort attribute must be searchable. That is, it must be a cataloged attribute in Oracle Internet Directory.

3.7 Understanding Paged LDAP Search Results

As of As of Oracle Internet Directory *g* (10.1.4.0.1), you can obtain sorted results from an LDAP search, as described by IETF RFC 2891.

You request sorted results by passing a control of type 1.2.840.113556.1.4.473 to the search function. The server returns a response control is of type 1.2.840.113556.1.4.474. Error processing and other details are described in RFC 2891. 10*g* (10.1.4.0.1), you can obtain paged results from an LDAP search, as described by IETF RFC 2696. You request sorted results by passing a control of type 1.2.840.113556.1.4.319 to the search function. Details are described in RFC 2696.

> **See Also:**
>
> IETF RFC 2696, "LDAP Control Extension for Simple Paged Results Manipulation," at http://www.ietf.org.

Sorting and paging may be used together.

The As of Oracle Internet Directory *g* (10.1.4.0.1), you can obtain sorted results from an LDAP search, as described by IETF RFC 2891. You request sorted results by passing a control of type 1.2.840.113556.1.4.473 to the search function. The server returns a response control is of type 1.2.840.113556.1.4.474. Error processing and other details are described in RFC 2891. implementation of RFC 2696 has the following limitations:

- The number of entries in a page might be less than the page size if an ACI partially blocks some entries from the search results.

- The paging response control does not contain the total entry count estimation. The return value is always 0.

3.8 Using Password Policies

The Oracle Internet Directory natively supports a rich set of policies governing passwords.

See Managing Password Policies in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*. You should design your applications to interact with the directory's password policies at runtime and handle any resulting events gracefully. Oracle Internet Directory provides several mechanisms to allow clients to interact with the server's password policies.

This section contains the following topics:

- Provisioning User
- Handling User Authentication
- About User Account Maintenance

3.8.1 Provisioning User

If your application provisions its own users in the directory, you should test passwords for acceptability by the server before committing the new user entry into the directory.

You can test a password by using the following custom control:

```
EffectivePolicyControl
::= SEQUENCE {
                        controlType     2.16.840.1.113894.1.8.29,
                        criticality     BOOLEAN DEFAULT FALSE,
                        controlValue    testPassword
                        }
testPassword ::= OCTET STRING (optional)
```

Package this control as part of an LDAP base search, where the base DN is that of the user entry being tested. The entry does not need to exist in the directory at the time. The server returns the LDAP entry describing the applicable password policy, assuming the entity performing the search has the access rights to view the password policy entry. Providing the desired password as the optional testPassword parameter results in the directory server returning the following response control:

```
PasswordValidationControl
::= SEQUENCE {
                        controlType     2.16.840.1.113894.1.8.32,
                        criticality     BOOLEAN DEFAULT FALSE,
                        controlValue    validationResult
                        }
validationResult ::= OCTET STRING
```

Your client application can parse the `validationResult` to determine whether the password is accepted by the server ("Success") or the reason it has been rejected. The error message is the same as that generated during a failed LDAP modify operation on userpassword.

3.8.2 Handling User Authentication

The user authentication process is performed within the directory server.

There are two main categories:

- Performing LDAP Bind/Compare Operation-Based Authentication
- Performing LDAP Search Operation-Based Authentication

3.8.2.1 Performing LDAP Bind/Compare Operation-Based Authentication

This refers to authentication performed on the standard `userpassword` attribute. The entire authentication process is performed within the directory server, and appropriate internal events are generated to update various account states as necessary.

The traditional use of this type of authentication is in an unsophisticated protocol. The application performs either an LDAP bind or compare operation against the server and checks for success. It handles all other cases as authentication failures.

When an authentication attempt fails, an application can determine the cause and either take action to remedy the situation or to expose the cause to the end user so that the end user can take an action to remedy the situation. This enhances the user experience and reduces administrative overhead. To retrieve such cause information during an LDAP Bind/Compare operation, you use the following LDAP control:

```
PasswordStatusRequestControl
::= SEQUENCE {
                      controlType     2.16.840.1.113894.1.8.6,
                      criticality     BOOLEAN DEFAULT FALSE,
                      }
```

When packaged as part of the LDAP Bind/Compare operation request, this control is processed by the server and causes the generation of a response control. The actual response control depends on the situation. See the password response controls in Table 3-3. Cases include imminent password expiration, number of grace logins remaining, password expired, and account locked.

3.8.2.2 Performing LDAP Search Operation-Based Authentication

This refers to authentication performed against other authentication tokens, such as password verifiers, maintained as part of a user entry. The token may be retrieved by the application. The authentication process occurs within the application and outside the scope of the directory server. Therefore in LDAP search-based authentications, the directory does not implicitly know the result of the authentication attempt.

If an application performs its own authentication after retrieving an authentication token from the directory, then none of the state-related policies are effective. Without these policies, scenarios such as locked accounts aren't enforceable, and users with expired accounts can still authenticate against the application.

The Oracle Internet Directory provides two mechanism that allow such an application to leverage the state related policy framework already present in the directory server. They are described in the following sections:

- Checking and Enforcing State Policies at Authentication Time
- Communicating to the Directory of Authentication Success/Failure

3.8.2.2.1 Checking and Enforcing State Policies at Authentication Time

You use the following custom control to enable this feature:

```
AccountStatusRequestControl
::= SEQUENCE {
                      controlType     2.16.840.1.113894.1.8.16,
```

```
                    criticality     BOOLEAN DEFAULT FALSE,
                  }
```

When this control is packaged with the LDAP search operation associated with the authentication process, Oracle Internet Directory processes this control and returns a response control to inform the client application of account state related information like account lockout, password expiration etc. See the password response controls in Table 3-3. The application can then parse and enforce the results.

While this addresses the issue of how the policies can be enforced by the client application, another fundamental application requirement is as follows:

3.8.2.2.2 Communicating to the Directory of Authentication Success/Failure

Oracle Internet Directory provides this ability through the virtual attribute: `orclAccountStatusEvent`.

This attribute is available on all user entries as a virtual attribute. That is, it has no disk footprint. However, modify operations can be applied on it. By default, a directory ships with restricted access to this attribute, so you must ask the directory administrator to grant your application identity write access on the attribute for the relevant user population.

You communicate authentication success or failure to the directory by modifying this attribute to hold *UserAccountEvent*. The following LDIF illustrates this.

```
dn: UserDN
changetype: modify
replace: orclAccountStatusEvent
orclAccountStatusEvent: UserAccountEvent
```

The Oracle Internet Directory understands the following values for *UserAccountEvent*:

```
UserAccountEvent = 1 (Authentication Success)
UserAccountEvent = 2 (Authentication Failure)
```

Upon receipt of these events, the Oracle Internet Directory invokes the same logic that is invoked during an authentication/success failure of an LDAP bind or compare operation, thereby updating the necessary account states.

In this way, you can leverage the existing account state related infrastructure available in the Oracle Internet Directory to secure your application.

3.8.3 About User Account Maintenance

User account maintenance and its interaction with password policies occur mostly around periodic password modifications. We recommend that applications utilize the EffectivePolicyControl described above to retrieve the effective policy and parse it to generate a message guiding the end user to the password construction requirements. Furthermore, we encourage the usage of the "test" capabilities encapsulated in this control to direct the end user towards the cause behind a modification failure. Handling these situations within the application reduces administrative overhead.

Secondly, there are a multitude of use-cases requiring a user to change his or her password upon next logon. The Oracle Internet Directory natively triggers this requirement when the pwdmustchange password policy element is enabled and a userpassword undergoes a non-self modification. However, in the event that an explicit

trigger of this requirement is needed, the Oracle Internet Directory supports it also via the orclAccountStatusEvent attribute described above. The relevant events are:

```
UserAccountEvent = 3 (Require Password Reset on next Logon)
UserAccountEvent = 4 (Clear Password Reset Requirement)
```

If the application has an administrative interface, this functionality may be desirable and can be exposed to the administrator.

4

Developing Applications With Oracle Extensions to the Standard APIs

Oracle extensions to the Java and PL/SQL LDAP APIs includes creating an application identity in the directory and understanding user management with LDAP APIs.

The following topics introduce the Oracle extensions to the Java and PL/SQL LDAP APIs:

- Understanding Oracle Extensions to the Standard APIs
- Creating an Application Identity in the Directory
- Features of User Management Using LDAP APIs
- Understanding Groups in Oracle Internet Directory
- Understanding Identity Management Realms
- Discovering a Directory Server

4.1 Understanding Oracle Extensions to the Standard APIs

You can use regular APIs with Oracle Extension APIs to achieve user management.

The following topics describe the API extensions to the standard API:

- API Extensions to the Standard API
- Steps Involved In Using API Extensions

4.1.1 API Extensions to the Standard API

The APIs that Oracle has added to the existing APIs to fulfill user, group and realm management.

The following are the functions:

Table 4-1   Functions and description

Functions	Description
User management	Applications can set or retrieve various user properties
Group management	Applications can query group properties
Realm management	Applications can set or retrieve properties about identity management realms
Server discovery management	Applications can locate a directory server in the Domain Name System (DNS)

Subsequent sections examine each of these functions in detail. Note that applications must use the underlying APIs for such common tasks as establishing and closing connections and looking up directory entries not searchable with the API extensions.

4.1.2 Steps Involved In Using API Extensions

The standard API functions and Oracle API extensions can be used when there is a connection established to Oracle Internet Directory.

Figure 4-1 shows what program flow looks like when the API extensions are used.

Figure 4-1   Programmatic Flow for API Extensions

As Figure 4-1 shows, an application first establishes a connection to Oracle Internet Directory. It can then use the standard API functions and the API extensions interchangeably.

Oracle does not support extensions to the C API.

4.2 Creating an Application Identity in the Directory

Before an application can use the LDAP APIs and their extensions, it must establish an LDAP connection.

After it establishes a connection, it must have permission to perform operations. But neither task can be completed if the application lacks an identity in the directory.

- Creating an Application Identity
- Assigning Privileges to an Application Identity

4.2.1 Creating an Application Identity

Creating an application identity in the directory is relatively simple. Such an entry requires only two object classes: `orclApplicationEntity` and `top`.

You can use either Oracle Directory Services Manager or an LDIF file to create the entry. In LDIF notation, the entry looks like this:

```
dn: orclapplicationcommonname=application_name
changetype: add
objectclass:top
```

```
objectclass: orclApplicationEntity
userpassword: password
```

The value provided for `userpassword` is the value that the application uses to bind to the directory.

4.2.2 Assigning Privileges to an Application Identity

After identifying the right set of privileges, you need to add the application entity DN to the appropriate directory groups.

To learn about the privileges available to an application, see Delegating Privileges for Oracle Identity Management in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*. The reference just provided explains how to perform this task using either Oracle Directory Services Manager or the `ldapmodify` command.

4.3 Features of User Management Using LDAP APIs

User management features of the LDAP APIs helps in retrieving user entries, extended user preferences etc.

Directory-enabled applications need to perform the following operations:

- Retrieve properties of user entries

  These properties are stored as attributes of the user entry itself—in the same way, for example, that a surname or a home address is stored.

- Retrieve extended user preferences

  These preferences apply to a user but are stored in a DIT different from the DIT containing user entries. Extended user preferences are either user properties common to all applications or user properties specific to an application. Those of the first type are stored in a common location in the Oracle Context. Those of the second type are stored in the application-specific DIT.

- Query the group membership of a user

- Authenticate a user given a simple name and credential

  Typically an application uses a fully qualified DN, GUID, or simple user name to identify a user. In a hosted environment, the application may use both a user name and a realm name for identification.

4.4 Understanding Groups in Oracle Internet Directory

Groups are modeled in Oracle Internet Directory as a collection of distinguished names. Directory-enabled applications must access Oracle Internet Directory to obtain the properties of a group and to verify that a given user is a member of that group.

A group is typically identified by one of the following:

- A fully qualified LDAP distinguished name

- A global unique identifier

- A simple group name along with a subscriber name

4.5 Understanding Identity Management Realms

An identity management realm is an entity or organization that subscribes to the services offered in the Oracle product stack. Directory-enabled applications must access Oracle Internet Directory to obtain realm properties such as user search base or password policy.

A realm is typically identified by one of the following:

- A fully qualified LDAP distinguished name
- A global unique identifier
- A simple enterprise name

4.6 Discovering a Directory Server

Directory server discovery (DSD) enables automatic discovery of the Oracle directory server by directory clients. It enables deployments to manage the directory host name and port number information in the central DNS server.

All directory clients perform a DNS query at runtime and connect to the directory server. Directory server location information is stored in a DNS service location record (SRV).

An SRV contains:

- The DNS name of the server providing LDAP service
- The port number of the corresponding port
- Any parameters that enable the client to choose an appropriate server from multiple servers

DSD also allows clients to discover the directory host name information from the `ldap.ora` file itself.

This section contains these topics:

- Benefits of Oracle Internet Directory Discovery Interfaces
- Creating Discovery Handle for Discovery Interfaces
- Determining Server Name and Port Number From DNS
- Environment Variables for DNS Server Discovery
- Overview of Programming Interfaces for DNS Server Discovery

> ✎ **See Also:**
>
> - "Discovering LDAP Services with DNS" by Michael P. Armijo at this URL:
>
>   http://www.ietf.org.
>
> - "A DNS RR for specifying the location of services (DNS SRV)", Internet RFC 2782 at the same URL.

4.6.1 Benefits of Oracle Internet Directory Discovery Interfaces

Typically, the LDAP host name and port information is provided statically in a file called `ldap.ora` which is located on the client in `$ORACLE_HOME/network/admin`.

For large deployments with many clients, this information becomes very cumbersome to manage. For example, each time the host name or port number of a directory server is changed, the `ldap.ora` file on each client must be modified.

Directory server discovery eliminates the need to manage the host name and port number in the `ldap.ora` file. Because the host name information resides on one central DNS server, the information must be updated only once. All clients can then discover the new host name information dynamically from the DNS when they connect to it.

DSD provides a single interface to obtain directory server information without regard to the mechanism or standard used to obtain it. Currently, Oracle directory server information can be obtained either from DNS or from `ldap.ora` using a single interface.

4.6.2 Creating Discovery Handle for Discovery Interfaces

The first step in discovering host name information is to create a discovery handle. A discovery handle specifies the source from which host name information is discovered. In case of the Java API, the discovery handle is created by creating an instance of the `oracle.ldap.util.discovery.DiscoveryHelper` class.

```
DiscoveryHelper disco = new DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);
```

The argument `DiscoveryHelper.DNS_DISCOVER` specifies the source. In this case the source is DNS.

Each source may require some inputs to be specified for discovery of host name information. In the case of DNS these inputs are:

- domain name
- discover method
- SSL mode

Detailed explanation of these options is given in "Determining Server Name and Port Number From DNS".

```
// Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN,"dc=us,dc=fiction,dc=com");
// Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD
   ,DiscoveryHelper.USE_INPUT_DN_METHOD);
// Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE,"0");
```

Now the information can be discovered.

```
// Call the discover method
disco.discover(reshdl);
```

The discovered information is returned in a result handle (`reshdl`). Now the results can be extracted from the result handle.

```
ArrayList result =
(ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);
if (result != null)
{
   if (result.size() == 0) return;
   System.out.println("The hostnames are :-");
   for (int i = 0; i< result.size();i++)
   {
      String host = (String)result.get(i);
      System.out.println((i+1)+".'"+host+"'");
   }
}
```

4.6.3 Determining Server Name and Port Number From DNS

Determining a host name and port number from a DNS lookup involves obtaining a domain and then searching for SRV resource records based on that domain. If there is more than one SRV resource record, they are sorted by weight and priority. The SRV resource records contain host names and port numbers required for connection. This information is retrieved from the resource records and returned to the user.

There are three approaches for determining the domain name required for lookup:

- Mapping the distinguished name (DN) of the naming context
- Using the domain component of local machine
- Looking up the default SRV record in the DNS

The following topics describe the preceding approaches:

- Using the DN of the Naming Context
- Using the Domain Component of Local Machine
- Using the Default SRV Record in DNS

4.6.3.1 Using the DN of the Naming Context

The first approach is to map the distinguished name (DN) of naming context into domain name using the algorithm given here.

The output domain name is initially empty. The DN is processed sequentially from right to left. An RDN is able to be converted if it meets the following conditions:

- It consists of a single attribute type and value
- The attribute type is `dc`
- The attribute value is non-null

If the RDN can be converted, then the attribute value is used as a domain name component (label).

The first such value becomes the right most, and the most significant, domain name component. Successive converted RDN values extend to the left. If an RDN cannot be converted, then processing stops. If the output domain name is empty when processing stops, then the DN cannot be converted into a domain name.

For the DN `cn=John Doe,ou=accounting,dc=example,dc=net`, the client converts the dc components into the DNS name `example.net`.

4.6.3.2 Using the Domain Component of Local Machine

Sometimes a DN cannot be mapped to a domain name. For example, the DN `o=Oracle IDC,Bangalore` cannot be mapped to a domain name. In this case, the second approach uses the domain component of the local machine on which the client is running. For example, if the client machine domain name is `mc1.example.com`, the domain name for the lookup is `example.com`.

4.6.3.3 Using the Default SRV Record in DNS

The third approach looks for a default SRV record in the DNS. This record points to the default server in the deployment. The domain component for this default record is `_default`.

After the domain name has been determined, it is used to send a query to DNS. The DNS is queried for SRV records specified in Oracle Internet Directory-specific format. For example, if the domain name obtained is `example.net`, the query for non-SSL LDAP servers is for SRV resource records having the owner name `_ldap._tcp._oid.example.net`.

It is possible that no SRV resource records are returned from the DNS. In such a case the DNS lookup is performed for the SRV resource records specified in standard format. For example, the owner name would be `_ldap._tcp.example.net`.

The result of the query is a set of SRV records. These records are then sorted and the host information is extracted from them. This information is then returned to the user.

> **Note:**
>
> The approaches mentioned here can also be tried in succession, stopping when the query lookup of DNS is successful. Try the approaches in the order as described in this section. DNS is queried only for SRV records in Oracle Internet Directory-specific format. If none of the approaches is successful, then all the approaches are tried again, but this time DNS is queried for SRV records in standard format.

4.6.4 Environment Variables for DNS Server Discovery

Environment variables override default behavior for discovering a DNS server.

The following table describes environment variables:

Table 4-2   Environment Variables for DNS Discovery

Environment Variable	Description
ORA_LDAP_DNS	IP address of the DNS server containing the SRV records. If the variable is not defined, then the DNS server address is obtained from the host machine.
ORA_LDAP_DNSPORT	Port number on which the DNS server listens for queries. If the variable is not defined, then the DNS server is assumed to be listening at standard port number 53.
ORA_LDAP_DOMAIN	Domain of the host machine. If the variable is not defined, then the domain is obtained from the host machine itself.

4.6.5 Overview of Programming Interfaces for DNS Server Discovery

The programming interface provided is a single interface to discover directory server information without regard to the mechanism or standard used to obtain it. Information can be discovered from various sources.

Each source can use its own mechanism to discover the information. For example, the LDAP host and port information can be discovered from the DNS acting as the source. Here DSD is used to discover host name information from the DNS.

> **✎ See Also:**
>
> For detailed reference information and class descriptions, refer to the Javadoc located on the product CD.

5

Using the Java API Extensions to JNDI

Java API extensions can be used with Java Naming and Directory Interface (JNDI) to perform the operations including authentication and creating dynamic password verifiers.

5.1 About Java Extension Installation

The Java extensions are installed along with the standard Java APIs when the LDAP client is installed.

The APIs and their extensions are found at `$ORACLE_HOME/jlib/ldapjclnt10.jar`.

5.2 Using the oracle.ldap.util Package to Model LDAP Objects

In Java, LDAP entities—users, groups, realms, and applications—are modeled as Java objects instead of as handles. This modeling is done in the `oracle.java.util` package. All other utility functionality is modeled either as individual objects—as, for example, `GUID`—or as static member functions of a utility class.

For example, to authenticate a user, an application must follow these steps:

1. Create `oracle.ldap.util.User` object, given the user DN.

2. Create a `DirContext` JNDI object with all of the required properties, or get one from a pool of `DirContext` objects.

3. Invoke the `User.authenticateUser` method, passing in a reference to the `DirContext` object and the user credentials.

4. If the `DirContext` object was retrieved from a pool of existing `DirContext` objects, return it to that pool.

Unlike their C and PL/SQL counterparts, Java programmers do not have to explicitly free objects. The Java garbage collection mechanism performs this task.

5.3 Using Classes PropertySetCollection, PropertySet, and Property

Many of the methods in the user, subscriber, and group classes return a `PropertySetCollection` object. The object represents a collection of one or more LDAP entries. Each of these entries is represented by a `PropertySet` object, identified by a DN.

A property set can contain attributes, each represented as a property. A property is a collection of one or more values for the particular attribute it represents. An example of the use of these classes follows:

```
PropertySetCollection psc = Util.getGroupMembership( ctx,
                                                     myuser,
                                                     null,
                                                     true );

   // for loop to go through each PropertySet
   for (int i = 0; i < psc.size(); i++ ) {

   PropertySet ps = psc.getPropertySet(i);

   // Print the DN of each PropertySet
   System.out.println("dn:  " + ps .getDN());

   // Get the values for the "objectclass" Property
   Property objectclass = ps.getProperty( "objectclass" );


   // for loop to go through each value of Property "objectclass"
   for (int j = 0; j< objectclass.size(); j++) {

       // Print each "objectclass" value
       System.out.println("objectclass:  " + objectclass.getValue(j));
   }
}
```

The entity `myuser` is a user object. The `psc` object contains all the nested groups that myuser belongs to. The code loops through the resulting entries and prints out all the object class values of each entry.

5.4 Managing Users Using oracle.ldap.util.User Java Class

All user-related functionality is abstracted in a Java class called `oracle.ldap.util.User`.

The process works like this:

1. Construct a `oracle.ldap.util.User` object based on a DN, GUID, or simple name.

2. Invoke `User.authenticateUser(DirContext, int, Object)` to authenticate the user if necessary.

3. Invoke `User.getProperties(DirContext)` to get the attributes of the user entry.

4. Invoke `User.getExtendedProperties(DirContext, int, String[])` to get the extended properties of the user. `int` is either shared or application-specific. `String[]` is the object that represents the type of property desired. If `String[]` is null, all properties in a given category are retrieved.

5. Invoke `PropertySetCollection.getProperties(int)` to get the metadata required to parse the properties returned in step 4.

6. Parse the extended properties and continue with application-specific logic. This parsing is also performed by application-specific logic.

5.5 Authenticating Users in LDAP operation

User authentication is a common LDAP operation that compares the credentials that a user provides at login with the user's credentials in the directory.

Oracle Internet Directory supports the following:

- Arbitrary attributes can be used during authentication

- Appropriate password policy exceptions are returned by the authentication method. Note, however, that the password policy applies only to the `userpassword` attribute.

The following code fragment shows how the API is used to authenticate a user:

```
 // User user1 - is a valid User Object
      try
      {
              user1.authenticateUser(ctx,
                  User.CREDTYPE_PASSWD, "welcome");

              // or
              // user1.authenticateUser(ctx, <any
attribute>, <attribute value>);
      }
      catch (UtilException ue)
      {
              // Handle the password policy error
accordingly
              if (ue instanceof PasswordExpiredException)
                    // do something
              else if (ue instanceof GraceLoginException)
                    // do something
      }
```

5.6 Creating Users Using createUser( )

The subscriber class uses the `createUser()` method to programmatically create users. The object classes required by a user entry are configurable through Oracle Delegated Administration Services.

The `createUser()` method assumes that the client understands the requirement and supplies the values for the mandatory attributes during user creation. If the programmer does not supply the required information the server returns an error.

The following snippet of sample code demonstrates the usage.

```
// Subscriber sub is a valid Subscriber object
// DirContext ctx is a valid DirContext

// Create ModPropertySet object to define all the attributes and their values.
ModPropertySet mps = new ModPropertySet();
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"cn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"sn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"mail",
"Anika@example.com");

// Create user by specifying the nickname and the ModPropertySet just defined
User newUser = sub.createUser( ctx, mps, true);

// Print the newly created user DN
System.out.println( newUser.getDN(ctx) );

// Perform other operations with this new user
```

5.7 Retrieving User Objects

The subscriber class offers the `getUser()` method to replace the public constructors of the User class. A user object is returned based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx is contains a valid directory connection with
sufficient privilege to perform the operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the default
subscriber
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_DEFAULT, null, null);

// Obtain a User object representing the user whose
nickname is "Anika"
User user1 = sub.getUser(ctx, Util.IDTYPE_SIMPLE, "Anika",
null);
// Do work with this user

The getUser() method can retrieve users based on DN, GUID
and simple name.  A getUsers() method is also available to
perform a filtered search to return more than one user at a
time.  The returned object is an array of User objects.
```

```
For example,

// Obtain an array of User object where the user's nickname
starts with "Ani"
User[] userArr = sub.getUsers(ctx, Util.IDTYPE_SIMPLE,
"Ani", null);
// Do work with the User array
```

5.8 Retrieving Objects from Realms

Java API can be used to retrieve objects in identity management realms. The `RootOracleContext` class represents the root Oracle Context. Much of the information needed for identity management realm creation is stored within the root Oracle Context.

The `RootOracleContext` class offers the `getSubscriber()` method. It replaces the public constructors of the subscriber class and returns an identity management realm object based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx contains a valid directory
// connection with sufficient privilege to perform the
// operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the
// Subscriber with simple name "Oracle"
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_SIMPLE, "Oracle", null);

// Do work with the Subscriber object
```

5.9 Searching for Oracle Single Sign-On Login Name

You can find a user's login name when you have the simple name, GUID, or DN. The Oracle Single Sign-On login name is also referred to as nickname.

There are two parts to this example:

1. Determine which attribute is used to store the nickname in this realm.

2. Retrieve the User object and determine the value of the nickname attribute.

```
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import oracle.ldap.util.jndi.*;
import oracle.ldap.util.*;
import java.io.*;

public class NickNameSearch {

    public static void main(String[] args)
        throws Exception
    {
        InitialLdapContext ctx = ConnectionUtil.getDefaultDirCtx( args[0],
                                  args[1], args[2],args[3]);
```

```
            RootOracleContext roc=new RootOracleContext(ctx);
            Subscriber sub = null;
            sub = roc.getSubscriber(ctx, Util.IDTYPE_DEFAULT, null, null) ;

            PropertySetCollection psc = sub.getProperties(ctx,
                                    Subscriber.USER_NAMING_PROPERTIES, null);

            String nickNameAttribute = null;
            try
            {
                nickNameAttribute = (String)
psc.getPropertySet(0).getProperty(Subscriber.USER_NAMING_ATTR_SIMPLE).getValue(0)
;
            }
            catch (Exception e)
            {
               // unable to retrieve the attribute name
               System.exit(0);
            }
            System.out.println("Nickname attribute: " + nickNameAttribute);

            // Retrieve user using simple name, guid or DN
            User user = sub.getUser(ctx, Util.IDTYPE_SIMPLE,"orcladmin", null);
            System.out.println("user DN: " + user.getDN(ctx));

            // Retrieve nickname value using User object
            psc = user.getProperties(ctx, new String[]{ nickNameAttribute });

            String nickName = null;
            try
            {
                nickName = (String)
psc.getPropertySet(0).getProperty(nickNameAttribute).getValue(0);
            }
            catch (Exception e)
            {
               // unable to retrieve the attribute value
               System.exit(0);
            }
            System.out.println("Nickname : " + nickName);
    }
}
```

5.10 Java Class Introduction for Discovering a Directory Server

The Java class `oracle.ldap.util.discovery.DiscoveryHelper` is used to discover a directory server from the specified source.

A new Java class, the public class, has been introduced:

```
public class oracle.ldap.util.discovery.DiscoveryHelper
```

This class provides a method for discovering specific information from the specified source.

Table 5-1   Methods for Directory Server Discovery

Method	Description
discover	Discovers the specific information from a given source
setProperty	Sets the properties required for discovery
getProperty	Accesses the value of properties

Two new methods are added to the existing Java class
`oracle.ldap.util.jndi.ConnectionUtil`:

- `getDefaultDirCtx`: This overloaded function determines the host name and port information of non-SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.

- `getSSLDirCtx`: This overloaded function determines the host name and port information of SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.

5.11 Discovering a Directory Server

A sample Java program is run for directory server discovery.

```
import java.util.*;
import java.lang.*;
import oracle.ldap.util.discovery.*;
import oracle.ldap.util.jndi.*;

public class dsdtest
{
  public static void main(String s[]) throws Exception
  {
    HashMap reshdl = new HashMap();
    String result = new String();
    Object resultObj = new Object();
    DiscoveryHelper disco = new
DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);

// Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN,"dc=us,dc=fiction,dc=com")
;

// Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD,
   DiscoveryHelper.USE_INPUT_DN_METHOD);

// Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE,"0");

// Call the discover method
int res=disco.discover(reshdl);
if (res!=0)
   System.out.println("Error Code returned by the discover method is :"+res) ;
```

```
// Print the results
printReshdl(reshdl);
}

public static void printReshdl(HashMap reshdl)
{
  ArrayList result = (ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);

if (result != null)
{
  if (result.size() == 0) return;
  System.out.println("The hostnames are :-");
  for (int i = 0; i< result.size();i++)
  {
      String host = (String)result.get(i);
      System.out.println((i+1)+".
'"+host+"'");
    }
  }
 }
}
```

5.12 Using DIGEST-MD5 to Perform SASL Authentication

When using JNDI to create a SASL connection, you must set these
`javax.naming.Context` properties.

- `Context.SECURITY_AUTHENTICATION = "DIGEST-MD5"`

- `Context.SECURITY_PRINCIPAL`

The latter sets the principal name. This name is a server-specific format. It can be
either of the following:

- The DN—that is, `dn:`—followed by the fully qualified DN of the entity being
  authenticated

- The string `u:` followed by the user identifier.

  The Oracle directory server accepts just a fully qualified DN such as
  `cn=user,ou=my department,o=my company.`

> **Note:**
>
> The SASL DN must be normalized before it is passed to the API that calls
> the SASL bind. To generate SASL verifiers, Oracle Internet Directory
> supports only normalized DNs.

5.13 Using SASL Digest-MD5 auth-int and auth-conf Modes

The following code provides an example of Java LDAP/JNDI using SASL Digest-MD5.

```
/* $Header: LdapSasl.java 27-oct-2005.11:26:59 qdinh Exp $ */

/* Copyright (c) 2003, 2005, Oracle. All rights reserved.  */
```

```
/*
  DESCRIPTION
   <short description of component this file declares/defines>

  PRIVATE CLASSES
   <list of private classes defined - with one-line descriptions>

  NOTES
   <other useful comments, qualifications, and so on.>

  MODIFIED    (MM/DD/YY)
   qdinh        04/23/03 - Creation
*/

/**
*  @version $Header: LdapSasl.java 27-oct-2005.11:26:59 qdinh Exp $
*  @author  qdinh  *  @since   release specific (what release of product did this
appear in)
*/

package oracle.ldap.util.jndi;

import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import oracle.ldap.util.jndi.*;
import oracle.ldap.util.*;
import java.lang.*;
import java.util.*;
public class LdapSasl
{
   public static void main( String[] args)
       throws Exception
   {

    int numofargs;

    numofargs = args.length;

    Hashtable hashtable = new Hashtable();

    // Look through System Properties for Context Factory if it is available
    // then set the CONTEXT factory only if it has not been set
    // in the environment -
    // set default to com.sun.jndi.ldap.LdapCtxFactory

    hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.ldap.LdapCtxFactory");
    // possible valid arguments
    // args[0] - hostname
    // args[1] - port number
    // args[2] - Entry DN
    // args[3] - Entry Password
    // args[4] - QoP [ auth | auth-int | auth-conf ]
    // args[5] - SASL Realm
    // args[6] - Cipher Choice
    //    If QoP == "auth-conf" then args[6] cipher choice can be
    //     - des
    //     - 3des
    //     - rc4
```

```
//     - rc4-56
//     - rc4-40

hashtable.put(Context.PROVIDER_URL, "ldap://"+args[0]+":"+args[1]);
    hashtable.put(Context.SECURITY_AUTHENTICATION, "DIGEST-MD5");
System.out.println("hash put security dn: " + args[2]);
hashtable.put(Context.SECURITY_PRINCIPAL, args[2] );
hashtable.put(Context.SECURITY_CREDENTIALS, args[3] );

// For Quality of Protection modes
// 1. Authentication and Data Integrity Mode - "auth-int"
// 2. Authentication and Data Confidentiality Mode  "auth-conf"

//
// hashtable.put("javax.security.sasl.qop",args[4]);
hashtable.put("javax.naming.security.sasl.realm", args[5]);

// Setup Quality of Protection
//
// System.out.println("hash sasl.qop: " + args[4]);

hashtable.put("javax.security.sasl.qop",args[4]);


if (numofargs > 4)
{
if (args[4].equalsIgnoreCase("AUTH-CONF"))
{

    //  Setup a cipher choice only if QoP == "auth-conf"
    String strength = "high";
    String cipher = new String(args[6]);
            if (cipher.compareToIgnoreCase("rc4-40") == 0)
    strength = "low";
    else if (cipher.compareToIgnoreCase("rc4-56") == 0 ||
        cipher.compareToIgnoreCase("des")== 0 )
    strength = "medium";
    else if (cipher.compareToIgnoreCase("3des") == 0 ||
        cipher.compareToIgnoreCase("rc4") == 0)
    strength = "high";

    // setup cipher choice
    System.out.println("hash sasl.strength:"+strength);
    hashtable.put("javax.security.sasl.strength",strength);
}

// set maxbuffer length if necessary
if (numofargs > 7 && !"".equals(args[6]))
    hashtable.put("javax.security.sasl.maxbuf", args[5].toString());
}


// Enable Debug --
// hashtable.put("com.sun.jndi.ldap.trace.ber", System.err);

LdapContext ctx = new InitialLdapContext(hashtable,null);


// At this stage - SASL Digest -MD5 has been successfully

System.out.println("sasl bind successful");
```

```
       // Ldap Search Scope Options
       //
       // - Search base - OBJECT_SCOPE
       // - One Level   - ONELEVEL_SCOPE
       // - Sub Tree    - SUBTREE_SCOPE
       //
          // Doing an LDAP Search
       PropertySetCollection psc =
Util.ldapSearch(ctx,"o=oracle,dc=com","objectclass=*",SearchControls.OBJECT_SCOPE,
                          new String[] {"*"});
         // Print out the serach result
       Util.printResults(psc);

       System.exit(0);

  }                                                                   }
```

5.14 Using LDAP Transactions

As of 11*g* Release 1 (11.1.1.6), theOracle Internet Directory SDK supports transactions, as defined in RFC 5805.Transactions are necessary to support a number of applications, including provisioning.

> **See Also:**
>
> RFC 5805 on Lightweight Directory Access Protocol (LDAP) Transactions at:
> http://www.ietf.org

When a transaction is committed, either all operations within the transaction update succeed or all fail. Oracle Internet Directory publishes the Object Identifiers of the extended operations and the control as supportedExtension and supportedControl in the root DSE.

• Entities Used in LDAP Transactions

• Sequence of Requests and Responses in LDAP Transactions

5.14.1 Entities Used in LDAP Transactions

Oracle Internet Directory LDAP transactions has the following entities for transaction request, response and end the transaction.

Table 5-2   LDAP transactions and entities

Entities	Description
Start Transaction Request (Object Identifier 1.3.6.1.1.21.1)	This is an LDAP extended operation that facilitates grouping of related operations for starting a transaction. It is an LDAPMessage of CHOICE extendedReq where requestName is 1.3.6.1.1.21.1 and the requestValue is absent.

Table 5-2   (Cont.) LDAP transactions and entities

Entities	Description
Start Transaction Response	This is sent by Oracle Internet Directory server in response to a Start Transaction Request. It is an LDAPMessage of CHOICE extendedRes. Its responseName is absent. Upon success, resultCode (0), responseValue is present and contains the transaction identifier. Upon failure, for example a malformed request, the responseValue is absent.
Transaction Specification Control (Object Identifier 1.3.6.1.1.21.2)	This is an LDAPControl indicating association of an operation to a transaction by means of the transaction identifier, which is the value of this control. Its criticality is TRUE.
End Transaction Request (Object Identifier 1.3.6.1.1.21.3)	This is another LDAP extended operation used to indicate the end of a transaction. This indicates the settling of the transaction, resulting in a commit or abort of the transaction. This LDAPMessage has a CHOICE extendedReq where the requestName is 1.3.6.1.1.21.3 and the requestValue is present and contains BER-encoded transaction request.
	A commit value of TRUE indicates a request to commit the transaction identified by the identifier. A commit value of FALSE indicates a request to abort the identified transaction.
End Transaction Response	This is an LDAPMessage response of Oracle Internet Directory server to an End Transaction Request. Its response name is absent. The responseValue, when present, contains a BER-encoded transaction response.
Aborted Transaction Notice (Object Identifier 1.3.6.1.1.21.4):	This is an Unsolicited Notification message where responseName is 1.3.6.1.1.21.4 and responseValue is present and contains a transaction identifier.

5.14.2 Sequence of Requests and Responses in LDAP Transactions

In an LDAP transaction, sequence of requests and responses occur between the Oracle Internet Directory server and its client.

The following is the sequence of requests and responses in an LDAP transaction:

1. A client issues a Start Transaction Request to Oracle Internet Directory server.

2. Oracle Internet Directory server responds to this request with a Start Transaction Response providing a transaction identifier and with a resultCode of success (0). Upon failure, for example, in the case of a malformed request, the server responds with a Start Transaction Response with a resultCode other than success indicating the nature of the error.

    This transaction identifier generated and sent to client in a success case is used in subsequent protocol messages to identify this transaction.

3. If the client receives a success result code from the server, it then attaches a Transaction Specification Control containing the transaction identifier to each of the update operations to indicate that they are to be processed as part of a single transaction. For a valid response, Oracle Internet Directory sends a success. Otherwise it sends to its client an appropriate response to the request with a resultCode along with a detailed nature of the failure.

4. After the client is done sending all the update operations of the transaction, accompanied by the Transaction Specification Control containing the transaction identifier to Oracle Internet Directory server, it sends an End Transaction Request including the transaction identifier to Oracle Internet Directory server, indicating that it wants to settle the transaction. A commit value of TRUE indicates a request to commit the transaction while a value of FALSE indicates a request to abort the transaction.

5. Upon receiving a request to abort the transaction, Oracle Internet Directory server aborts the identified transaction by abandoning all the operations that were part of the transaction and indicates that it has done so by returning an End Transaction Response with a resultCode of success (0). Upon receiving a request to commit the transaction, Oracle Internet Directory server applies commit to DB for the updates it performed for all the operations in the transaction. If it succeeds, the server returns an End Transaction Response with a resultCode of success (0). Otherwise, it returns an End Transaction Response with a non-success resultCode, indicating the nature of the failure.

   There is no requirement that the server serialize transactions or updates requested outside of the transaction. That is, the server may process multiple commit requests, from one or more clients, acting upon different sets of entries concurrently.

6. If, at any time during the above exchange between the client and server, the server is unable to continue the specification of a transaction, the server issues and Aborted Transaction Notice. Upon receiving this notice, the client must discontinue all use of the transaction identifier and the transaction is null and void. All operations that were to be processed as part of the transaction are implicitly abandoned. Any further use of the identifier by the client results in a response containing a failure resultCode from Oracle Internet Directory server.

7. If, at any time during the above exchange between the client and server, the client closes the connection to Oracle Internet Directory server, the server performs the necessary connection-related clean up.

5.15 Using JNDI-Based LDAP Transactions

You will learn the steps required to develop a JNDI-based LDAP transaction Java client module using LDAP extended operations.The steps include implementing four interfaces: Start Transaction Request, Start Transaction Response, End Transaction Request, and End Transaction Response.

The classes implementing these four interfaces must be used in code performing LDAP update operations within transaction semantics.

The following sample code snippets show the four implemented interfaces and their use in a sample LDAP update operations with transaction semantics:

- Implementing Start Transaction Request Interface (LdapStartTxnReq.java)
- Implementing Start Transaction Response Interface (LdapStartTxnRes.java)
- Implementing End Transaction Request Interface (LdapEndTxnReq.java)
- Implementing End Transaction Response Interface (LdapEndTxnRes.java)
- Using Java and Pseudocode Sample of Interfaces in LDAP Update Operations Within Transaction Semantics (LdapTxnOperation.java)

5.15.1 Implementing Start Transaction Request Interface (LdapStartTxnReq.java)

You can implement Start Transaction Request Interface by using the Java class `LdapStartTxnReq`.

The following code shows start transaction request interface implementation:

```
import java.lang.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;

public class LdapStartTxnReq implements ExtendedRequest
{

    public static final String TXN_REQ_OID = "1.3.6.1.1.21.1";

    // constructor
    public LdapStartTxnReq()
    {
    }

    // Method for getting the request ID
    public String getID() {
      return TXN_REQ_OID;
    }

    public byte[] getEncodedValue() {
      return null;  // No value is needed for Start Txn request
    }

    public ExtendedResponse createExtendedResponse(String id, byte[] berValue,
                                                  int offset, int length)
          throws NamingException

    {
        return new LdapStartTxnRes(id, berValue, offset, length);     }
}
```

5.15.2 Implementing Start Transaction Response Interface (LdapStartTxnRes.java)

You can implement Start Transaction Response Interface by using the Java class `LdapStartTxnRes`.

The following code shows start transaction response interface:

```
import java.lang.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
```

```java
public class LdapStartTxnRes implements ExtendedResponse
{
  public String txnID;

  // Called by LdapStartTxnReq.createExtendedResponse()
  protected LdapStartTxnRes(String id, byte[] berValue, int offset, int length)
            throws NamingException
  {
    byte[] buff2 = new byte[length];

    System.arraycopy(berValue, offset, buff2, 0, length);

    try
    {
        this.txnID = new String(berValue, "UTF8");
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }

  public String getTxnID()
  {
    return txnID;
  }

  // No op for our case
  public byte[] getEncodedValue()
  {
    return null;
  }

  public String getID()
  {
    return null; // as have no response ID.
  }
}
```

5.15.3 Implementing End Transaction Request Interface (LdapEndTxnReq.java)

You can implement End Transaction Request Interface by using the Java class
`LdapEndTxnReq`.

The following code shows end transaction request interface implementation:

```java
import java.lang.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.ldap.*;
import oracle.ldap.util.ber.*;


public class LdapEndTxnReq implements ExtendedRequest
{
```

```
public  static final String TXN_END_REQ_OID = "1.3.6.1.1.21.3";
private int sequenceTag = 0x30;
private boolean commitValue;
private String  controlValue;

// constructor
public LdapEndTxnReq(boolean commitVal, String controlVal)
{
  this.commitValue = commitVal;
  this.controlValue = controlVal;
}

// Method to get End Txn Request ID
public String getID() {
    return TXN_END_REQ_OID;
}

public byte[] getEncodedValue()
{
  boolean encodeUTF8 = true;
  byte[] bytes = null;

  try
  {
    BEREncoder berElement = new BEREncoder();

    berElement.beginSeq(sequenceTag);

    berElement.encodeBoolean(this.commitValue);

    berElement.encodeString(this.controlValue, encodeUTF8);

    berElement.endSeq();
    bytes = berElement.getTrimmedBuf();

    return bytes;
  }
  catch (EncodeException e)
  {
     return null;
  }
}

public ExtendedResponse createExtendedResponse(String id, byte[] berValue,
                                              int offset, int length)
      throws NamingException
{
  return null;
}
}
```

5.15.4 Implementing End Transaction Response Interface (LdapEndTxnRes.java)

You can implement End Transaction Response Interface by Java class
LdapEndTxnRes.

The following code shows end transaction response interface implementation:

```
import java.lang.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;


public class LdapEndTxnRes implements ExtendedResponse
{
  // constructor
  LdapEndTxnRes(String id, byte[] berValue, int offset, int length)
               throws NamingException
  {
    ;
  }

  public String getTxnID()
  {
    return null;
  }

  public byte[] getEncodedValue()
  {
    return null;
  }

  public String getID()
  {
    return null;
  }
}
```

5.15.5 Using Java and Pseudocode Sample of Interfaces in LDAP Update Operations Within Transaction Semantics (LdapTxnOperation.java)

To perform LDAP Update operations you can use Java and pseudocode interface.

The following code shows java and pseudocode sample of interfaces:

```
import java.lang.*;
import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
                   .
                   .
                   .
public class LdapTxnOperation implements  …………….
{
    public static void main( String[] args )
    {
        try
        {
            LdapContext  ctx;

            Perform ldapbind against OID server on the LDAP context ctx;
```

```
             If  bind is successful
             {
                 /* Issue LDAP Start Transaction Request to OID server and
                    receive its response */
                 LdapStartTxnRes startResp =
(LdapStartTxnRes)ctx.extendedOperation(new

   LdapStartTxnReq());
                 /* Extract the transaction ID sent by OID server */
                 String txnID = startResp.getTxnID();

Issue LDAP update operations (add, delete, modify) to OID server each containing
 the Transaction Control with ID 1.3.6.1.1.21.2 and value of the transaction ID;

/* Vendor's technical doc link is provided in Reference section below */

/* After done sending all update operations, issue LDAP End Transaction Request
 to OID server with your intention to either commit or abort the transaction and
 receive its response. */

If  want to commit the transaction
{ /* commit request */
    LdapEndTxnRes endResp = (LdapEndTxnRes)ctx.extendedOperation(new
LdapEndTxnReq(true, txnID));
 }
 Else
 { /* abort request */
    LdapEndTxnRes endResp = (LdapEndTxnRes)ctx.extendedOperation(new
LdapEndTxnReq(false, txnID));
 }
               }
               catch( Exception e)
             {
                e.printStackTrace();
                return;
             }
             return ;
        }
      }
```

6
Using the API Extensions in PL/SQL

PL/SQL extensions are used with standard directory APIs to manage and authenticate users using `DBMS_LDAP_UTL.authenticate_user()` function.

The following topics explain how to use PL/SQL extensions to the standard directory APIs to manage and authenticate users:

- Installing the PL/SQL Extensions
- Process Flow to Create User Handles to Access Directory Data
- Process Flow to Use the DBMS_LDAP_UTL Package
- Authenticating Users using DBMS_LDAP_UTL.authenticate_user()
- Dependencies and Limitations of the PL/SQL LDAP API

> **Note:**
>
> The Oracle extensions do not include PL/SQL APIs that create users. The Oracle extensions to the standard APIs are documented in full in DBMS_LDAP_UTL PL/SQL Reference.

6.1 Installing the PL/SQL Extensions

The PL/SQL extensions are installed with the `DBMS_LDAP` package when the Oracle database is installed.

You must run the script `$ORACLE_HOME/rdbms/admin/catldap.sql`.

6.2 Process Flow to Create User Handles to Access Directory Data

Most of the extensions described in this chapter are helper functions. They access data about specific LDAP entities such as users, groups, realms, and applications.

In many cases, these functions must pass a reference to one of these entities to the standard API functions. To do this, the API extensions use opaque data structures called handles. The steps that follow show an extension creating a user handle:

1. Establish an LDAP connection or get one from a pool of connections.
2. Create a user handle from user input. This could be a DN, a GUID, or a single sign-on user ID.
3. Authenticate the user with the LDAP connection handle, user handle, or credentials.
4. Free the user handle.

5. Close the LDAP connection, or return the connection back to the connection pool.

6.3 Process Flow to Use the DBMS_LDAP_UTL Package

The steps that follow show how the `DBMS_LDAP_UTL` package is used to create and use a handle that retrieves user properties from the directory.

1. Invoke `DBMS_LDAP_UTL.create_user_handle(user_hd, user_type, user_id)` to create a user handle from user input. The input can be a DN, a GUID, or a single sign-on user ID.

2. Invoke `DBMS_LDAP_UTL.set_user_handle_properties(user_hd, property_type, property)` to associate a realm with the user handle.

3. Invoke `DBMS_LDAP_UTL.get_user_properties(ld, user_handle, attrs, ptype, ret_pset_coll)` to place the attributes of a user entry into a result handle.

4. Invoke `DBMS_LDAP_UTL.get_property_names(pset, property_names)` and `DBMS_LDAP_UTL.get_property_values(pset, property_name, property_values)` to extract user attributes from the result handle that you obtained in step 3.

6.4 Authenticating Users using DBMS_LDAP_UTL.authenticate_user()

Use `DBMS_LDAP_UTL.authenticate_user(session, user_handle, auth_type, cred, binary_cred)` to authenticate a user to the directory.

This function compares the password provided by the user with the password attribute in the user's directory entry.

6.5 Dependencies and Limitations of the PL/SQL LDAP API

The PL/SQL LDAP API for this release has limitations for database session.

Some of the limitations are described below:

- The LDAP session handles obtained from the API are valid only for the duration of the database session. The LDAP session handles cannot be written to a table and reused in other database sessions.

- Only synchronous versions of LDAP API functions are supported in this release.

  The PL/SQL LDAP API requires a database connection to work. It cannot be used in client-side PL/SQL engines (like Oracle Forms) without a valid database connection.

7

Developing Provisioning-Integrated Applications

In the 10g (10.1.4.0.1) release, new APIs were added for developing provisioning-integrated applications.

Nothing has changed for these new APIs since 11g Release 1 (11.1.1). For more information, see Deploying Provisioning-Integrated Applications in *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*.

Part II

Oracle Internet Directory Programming Reference

You can use the standard APIs such as C API, DBMS LDAP PL/SQL, JAVA API, and Oracle extensions to these APIs.

- C API Reference
- DBMS_LDAP PL/SQL Reference
- Java API Reference
- DBMS_LDAP_UTL PL/SQL Reference
- Oracle Directory Integration and Provisioning Java API Reference
- Oracle Directory Integration Platform PL/SQL API Reference

8

C API Reference

Oracle Internet Directory C API uses Oracle extensions to support SSL.
The following topics describe Oracle Internet Directory C API and provides examples of how to use this API:

- Overview of Oracle Internet Directory C API
- C API Functions at a Glance
- Sample C API Usage
- Implementing Header Files and Libraries for the C API Usage
- Dependencies and Limitations of the C API

8.1 Overview of Oracle Internet Directory C API

The Oracle Internet Directory SDK C API is based on LDAP Version 3 C API and Oracle extensions to support SSL.

Oracle Internet Directory C API is described in the following topics:

- Understanding the Modes of Oracle Internet Directory
- Understanding Oracle Internet Directory SDK C API SSL Extensions

8.1.1 Understanding the Modes of Oracle Internet Directory

You can use the Oracle Internet Directory API in SSL and non-SSL modes.

The following modes are described below:

- SSL—All communication secured by using SSL
- Non-SSL—Client/server communication not secure

The API uses TCP/IP to connect to a directory server. When it does this, it uses, by default, an unencrypted channel. To use the SSL mode, you must use the Oracle SSL call interface. You determine which mode you are using by the presence or absence of the SSL calls in the API usage. You can easily switch between SSL and non-SSL modes.

> ✏️ **See Also:**
>
> Sample C API Usage for more details on how to use the two modes.

8.1.2 Understanding Oracle Internet Directory SDK C API SSL Extensions

Oracle SSL extensions to the LDAP API are based on standard SSL protocol. The SSL extensions provide encryption and decryption of data over the wire and authentication.

There are three modes of authentication:

- None—Neither client nor server is authenticated, and only SSL encryption is used
- One-way—Only the server is authenticated by the client
- Two-way—Both the server and the client are authenticated by each other

The type of authentication is indicated by a parameter in the SSL interface call. The following sections describes it briefly:

- Interface Calls to Enable SSL
- About Wallet Support

8.1.2.1 Interface Calls to Enable SSL

There is only one call required to enable SSL:

```
int ldap_init_SSL(Sockbuf *sb, char *sslwallet, char *sslwalletpasswd, int
sslauthmode)
```

The `ldap_init_SSL` call performs the necessary handshake between client and server using the standard SSL protocol. If the call is successful, then all subsequent communication happens over a secure connection.

Table 8-1   Arguments for SSL Interface Calls

Argument	Description
sb	Socket buffer handle returned by the `ldap_open` call as part of LDAP handle.
sslwallet	Location of the user wallet.
sslwalletpasswd	Password required to use the wallet.
sslauthmode	SSL authentication mode user wants to use. Possible values are: • `GSLC_SSL_NO_AUTH`—No authentication required • `GSLC_SSL_ONEWAY_AUTH`—Only server authentication required. • `GSLC_SSL_TWOWAY_AUTH`—Both server and client authentication required. A return value of `0` indicates success. A nonzero return value indicates an error. The error code can be decoded by using the function `ldap_err2string`.

> ✎ **See Also:**
>
> Sample C API Usage.

8.1.2.2 About Wallet Support

Depending on which authentication mode is being used, both the server and the client may require wallets to use the SSL feature. The API only has support for Oracle Wallet. You can create wallets by using Oracle Wallet Manager.

8.2 C API Functions at a Glance

C API functions uses many procedures to perform different operations in the directory.

Table 8-2 lists all of the functions and procedures in the C API and briefly explains their purpose.

Table 8-2   Functions and Procedures in the C API

Function or Procedure	Description
`ber_free`	Free the memory allocated for a BerElement structure
`ldap_abandon_ext` `ldap_abandon`	Cancel an asynchronous operation

`ldap_add_ext`
`ldap_add_ext_s`
`ldap_add`
`ldap_add_s`
	Add a new entry to the directory

`ldap_compare_ext`
`ldap_compare_ext_s`
`ldap_compare`
`ldap_compare_s`
	Compare entries in the directory

`ldap_count_entries`	Count the number of entries in a chain of search results
`ldap_count_values`	Count the string values of an attribute
`ldap_count_values_len`	Count the binary values of an attribute
`ora_ldap_create_clientctx`	Create a client context and returns a handle to it.
`ora_ldap_create_cred_hdl`	Create a credential handle.

`ldap_delete_ext`
`ldap_delete_ext_s`
`ldap_delete`
`ldap_delete_s`
	Delete an entry from the directory

`ora_ldap_destroy_clientctx`	Destroy the client context.

Table 8-2   (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
`ora_ldap_free_cred_hdl`	Destroy the credential handle.
`ldap_dn2ufn`	Converts the name into a more user friendly format
`ldap_err2string`	Get the error message for a specific error code
`ldap_explode_dn`	Split up a distinguished name into its components
`ldap_explode_rdn`	
`ldap_first_attribute`	Get the name of the first attribute in an entry
`ldap_first_entry`	Get the first entry in a chain of search results
`ora_ldap_get_cred_props`	Retrieve properties associated with credential handle.
`ldap_get_dn`	Get the distinguished name for an entry
`ldap_get_option`	Access the current value of various session-wide parameters
`ldap_get_values`	Get the string values of an attribute
`ldap_get_values_len`	Get the binary values of an attribute
`ldap_init` `ldap_open`	Open a connection to an LDAP server
`ora_ldap_init_SASL`	Perform SASL authentication
`ldap_memfree`	Free memory allocated by an LDAP API function call

`ldap_modify_ext`
`ldap_modify_ext_s`
`ldap_modify`
`ldap_modify_s`
	Modify an entry in the directory

`ldap_msgfree`	Free the memory allocated for search results or other LDAP operation results
`ldap_first_attribute` `ldap_next_attribute`	Get the name of the next attribute in an entry

Table 8-2   (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
`ldap_next_entry`	Get the next entry in a chain of search results
`ldap_perror` (Deprecated)	Prints the message supplied in message.
`ldap_rename` `ldap_rename_s`	Modify the RDN of an entry in the directory
`ldap_result2error` (Deprecated)	Return the error code from result message.

`ldap_result`
`ldap_msgfree`
`ldap_msgtype`
`ldap_msgid`
	Check the results of an asynchronous operation

`ldap_sasl_bind` `ldap_sasl_bind_s`	General authentication to an LDAP server

`ldap_search_ext`
`ldap_search_ext_s`
`ldap_search`
`ldap_search_s`
	Search the directory

`ldap_search_st`	Search the directory with a timeout value
`ldap_get_option` `ldap_set_option`	Set the value of these parameters
`ldap_set_rebind_proc`	Set the callback function to be used to get bind credential to a new server when chasing referrals.
`ora_ldap_set_clientctx`	Add properties to the client context handle.
`ora_ldap_set_cred_props`	Add properties to credential handle.

`ldap_simple_bind`
`ldap_simple_bind_s`
`ldap_sasl_bind`
`ldap_sasl_bind_s`
	Simple authentication to an LDAP server

`ldap_unbind_ext`
`ldap_unbind`
`ldap_unbind_s`
	End an LDAP session

Table 8-2   (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
`ldap_value_free`	Free the memory allocated for the string values of an attribute
`ldap_value_free` `ldap_value_free_len`	Free the memory allocated for the binary values of an attribute

This section lists all the calls available in the LDAP C API found in RFC 1823.

> ✎ **See Also:**
>
> The following URL for a more detailed explanation of these calls:
>
> http://www.ietf.org

8.3 Sample C API Usage

The first three examples show how to use the C API both with and without SSL and for SASL authentication. More complete examples are given in RFC 1823.The sample code for the command-line tool to perform an LDAP search also demonstrates use of the API in both the SSL and the non-SSL mode.

C API usage is explained in these topics:

- Using C API with SSL

- Using C API Without SSL

- Using C API for SASL-Based DIGEST-MD5 Authentication

- Setting and Using a Callback Function to Get Credentials When Chasing Referrals

8.3.1 Using C API with SSL

If the user makes `ldap_init_SSL` call the client/server communication is secured by using SSL.

The following code describes C API Usage with SSL:

```
#include <stdio.h>
#include <ldap.h>

main()
{

LDAP            *ld;
int                 ret = 0;
….
/* open a connection */
if ((ld = ldap_open("MyHost", 3131)) == NULL)
     exit( 1 );
```

```
/* SSL initialization */
ret = ldap_init_SSL(&ld->ld_sb, "file:/sslwallet", "welcome",
                                        GSLC_SSL_ONEWAY_AUTH );
if(ret != 0)
{

printf(" %s \n", ldap_err2string(ret));
exit(1);

}

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_bind_s" );
    exit( 1 );
}


.
.
.

}
```

8.3.2 Using C API Without SSL

If the user is not making the `ldap_init_SSL` call, and the client-to-server communication is therefore not secure.

The following code describes C API usage without SSL:

```
#include <stdio.h>
#include <ldap.h>

main()
{

LDAP            *ld;
int             ret = 0;
.
.
.
/* open a connection */
if ( (ld = ldap_open( "MyHost", LDAP_PORT )) == NULL )
    exit( 1 );

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_bind_s" );
     exit( 1 );
}
.
.
.

}
```

8.3.3 Using C API for SASL-Based DIGEST-MD5 Authentication

LDAP SASL C-API is used for SASL-based DIGEST-MD5 authentication to a directory server.

This sample program illustrates the usage of LDAP SASL C-API:

```
/*
   EXPORT FUNCTION(S)
     NONE

   INTERNAL FUNCTION(S)
     NONE

   STATIC FUNCTION(S)
     NONE

   NOTES
     Usage:
      saslbind -h ldap_host -p ldap_port -D authentication_identity_dn \
               -w password
      options
       -h      LDAP host
       -p      LDAP port
       -D      DN of the identity for authentication
       -p      Password

      Default SASL authentication parameters used by the demo program
      SASL Security Property :    Currently only "auth" security property
                                  is supported by the C-API. This demo
                                  program uses this security property.
      SASL Mechanism         :    Supported mechanisms by OID
                                  "DIGEST-MD5" - This demo program
                                                 illustrates it's usage.
                                  "EXTERNAL" - SSL authentication is used.
                                               (This demo program does
                                                not illustrate it's usage.)
      Authorization identity :    This demo program does not use any
                                   authorization identity.

   MODIFIED    (MM/DD/YY)
    ******       06/12/03 - Creation

*/

/*---------------------------------------------------------------------------
                     PRIVATE TYPES AND CONSTANTS
   ------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
                     STATIC FUNCTION DECLARATIONS
   ------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <ldap.h>


static int ldap_version = LDAP_VERSION3;
```

```
main (int argc, char **argv)
{
  LDAP*           ld;
  extern char*    optarg;
  char*           ldap_host = NULL;
  char*           ldap_bind_dn = NULL;
  char*           ldap_bind_pw = NULL;
  int             authmethod = 0;
  char            ldap_local_host[256] = "localhost";
  int             ldap_port = 3060;
  char*           authcid = (char *)NULL;
  char*           mech = "DIGEST-MD5"; /* SASL mechanism */
  char*           authzid = (char *)NULL;
  char*           sasl_secprops = "auth";
  char*           realm = (char *)NULL;
  int             status = LDAP_SUCCESS;
  OraLdapHandle   sasl_cred = (OraLdapHandle )NULL;
  OraLdapClientCtx *cctx = (OraLdapClientCtx *)NULL;
  int              i = 0;


    while (( i = getopt( argc, argv,
        "D:h:p:w:E:P:U:V:W:O:R:X:Y:Z"
        )) != EOF ) {
    switch( i ) {

    case 'h':    /* ldap host */
        ldap_host = (char *)strdup( optarg );
        break;
    case 'D':    /* bind DN */
        authcid = (char *)strdup( optarg );
        break;

    case 'p':    /* ldap port */
        ldap_port = atoi( optarg );
        break;
    case 'w':    /* Password */
        ldap_bind_pw = (char *)strdup( optarg );
        break;

    default:
        printf("Invalid Arguments passed\n" );
    }
    }


  /* Get the connection to the LDAP server */
  if (ldap_host == NULL)
    ldap_host = ldap_local_host;

  if ((ld = ldap_open (ldap_host, ldap_port)) == NULL)
  {
    ldap_perror (ld, "ldap_init");
    exit (1);
  }

  /* Create the client context needed by LDAP C-API Oracle Extension functions*/
  status = ora_ldap_init_clientctx(&cctx);

  if(LDAP_SUCCESS != status) {
     printf("Failed during creation of client context \n");
```

```
        exit(1);
    }

    /* Create SASL credentials */
    sasl_cred = ora_ldap_create_cred_hdl(cctx, ORA_LDAP_CRED_HANDLE_SASL_MD5);

    ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_REALM,
        (void *)realm);
    ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTH_PASSWORD,
        (void *)ldap_bind_pw);
    ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTHORIZATION_ID,
        (void *)authzid);
    ora_ldap_set_cred_props(cctx, sasl_cred,
ORA_LDAP_CRED_SASL_SECURITY_PROPERTIES,
        (void *)sasl_secprops);

    /* If connecting to the directory using SASL DIGEST-MD5, the Authentication ID
       has to be normalized before it's sent to the server,
       the LDAP C-API does this normalization based on the following flag set in
       SASL credential properties */
    ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_NORM_AUTHDN, (void
*)NULL);

    /* SASL Authetication to LDAP Server */
    status = (int)ora_ldap_init_SASL(cctx, ld, (char *)authcid, (char
*)ORA_LDAP_SASL_MECH_DIGEST_MD5,
                        sasl_cred, NULL, NULL);

    if(LDAP_SUCCESS == status) {
        printf("SASL bind successful \n" );
    }else {
        printf("SASL bind failed with status :  %d\n", status);
    }

    /* Free SASL Credentials */
    ora_ldap_free_cred_hdl(cctx, sasl_cred);

    status = ora_ldap_free_clientctx(cctx);

    /* Unbind from LDAP server */
    ldap_unbind (ld);

    return (0);
}


/* end of file saslbind.c */
```

8.3.4 Setting and Using a Callback Function to Get Credentials When Chasing Referrals

To set the callback function, you use ldap_set_rebind_proc(). The callback function is used only if LDAP_OPT_REFERRALS is set using ldap_set_option(). If ldap_set_rebind_proc() is not called, then the library uses anonymous bind to connect to a new server while chasing LDAP referrals.

```
/* referralsample.c - Sample program to demonstrate the usage of
ldap_set_rebind_proc() for referrals */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <ldap.h>


/*
* Prints the Entry DNs of the search result
*/
void print_entry_dns( LDAP *ld, LDAPMessage *result )
{
    LDAPMessage      *e;
    char             *dn;

    for ( e = ldap_first_entry( ld, result ); e != NULL;
        e = ldap_next_entry(ld, e ) )
    {
        if ( (dn = ldap_get_dn( ld, e )) != NULL ) {
            printf( "dn: %s\n\n", dn );
            ldap_memfree( dn );
        }
        else {
            ldap_perror( ld, "ldap_get_dn" );
        }
    }
}

/*
* Rebind function for providing the credentials to bind referral servers
*/
int getbindcredentials(LDAP *ld, char **binddn, char **bindpwd, int *authmethod, int
freeit)
{
    if (freeit == 0) {
    /* In this example bind credentials are static.  Typically, Bind credentials are
fetched from wallet or
some other means for the server information in input session handle */

        *binddn = "cn=orcladmin";
        *bindpwd = "xyz";
        *authmethod = LDAP_AUTH_SIMPLE;
    }
    else {
    /* In this example there is no memory allocation.
        If the memory is allocated for binddn/bindpwd/authmehod, they should be freed
here */

        *binddn = NULL;
        *bindpwd = NULL;
        *authmethod = 0;
    }

    return 0;
}

main()
{
    char             ldaphost[] = "localhost";
    char             binddn[] = "cn=orcladmin";
    char             bindpwd[] = "password";
    int              ldapport = 3060;
    char             searchbase[] = "dc=oracle,dc=com";
```

```
char            filter[] = "objectclass=*";
int             scope = LDAP_SCOPE_SUBTREE;

LDAP           *ld;
LDAPMessage    *result;
int             ret = 0;


if ( (ld = ldap_open( ldaphost, ldapport )) == NULL) {
    printf( "ldap_open: Connection failed\n" );
    exit( 1 );
}

if ( ldap_simple_bind_s(ld, binddn, bindpwd) != LDAP_SUCCESS ) {
    ldap_perror(ld, "ldap_simple_bind_s");
    exit( 1 );
}

/* Set this option to connect to the referrals */
ldap_set_option (ld, LDAP_OPT_REFERRALS, (void *)1);

/* set the function pointer which provides the bind credentials for referral
server */
ldap_set_rebind_proc(ld,  (int (*)(LDAP*, char**, char**, int*,
int))getbindcredentials);


ret = ldap_search_s( ld, searchbase, scope, filter, NULL, 0, &result );
if(LDAP_SUCCESS != ret) {
  ldap_perror(ld, "ldap_search_s");
  exit( 1 );
}

print_entry_dns(ld, result);

ldap_unbind(ld);
return(0);
}
```

8.4 Implementing Header Files and Libraries for the C API Usage

To build applications with the C API, you need to link it to the library.

- Include the header file located at `$ORACLE_HOME/ldap/public/ldap.h`
- Dynamically link to the library located at:
    - `$ORACLE_HOME/lib/libclntsh.so.11.1` on UNIX and Linux operating systems
    - `%ORACLE_HOME%\bin\oraldapclnt11.lib` on Windows operating systems

8.5 Dependencies and Limitations of the C API

This API can work against any release of Oracle Internet Directory. It requires either an Oracle environment or, at minimum, globalization support and other core libraries.

To use the different authentication modes in SSL, the directory server requires corresponding configuration settings.

> **✎ See Also:**
>
> Configuring Secure Sockets Layer (SSL) in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for details about how to set the directory server in various SSL authentication modes.

Oracle Wallet Manager is required for creating wallets if you are using the C API in SSL mode.

TCP/IP Socket Library is required.

The following Oracle libraries are required:

- Oracle SSL-related libraries
- Oracle system libraries

Sample libraries are included in the release for the sample command line tool. You should replace these libraries with your own versions of the libraries.

The product supports only those authentication mechanisms described in LDAP SDK specifications (RFC 1823).

All strings input to the C API must be in UTF-8 format. If the strings are not in the UTF-8 format, you can use the OCI function `OCINlsCharSetConvert` to perform the conversion. Please see the *Oracle Call Interface Programmer's Guide* in the Oracle Database Library at `http://www.oracle.com/technology/documentation`.

8.6 Initializing an LDAP Session and Usage Notes

The calls initialize a session with an LDAP server.

LDAP session and usage notes are described in the following sections:

- ldap_init and ldap_open

8.6.1 ldap_init and ldap_open

`ldap_init()` initializes a session with an LDAP server, but does not open a connection. The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization. `ldap_open()` initializes a session and opens a connection. The two fulfill the same purpose and have the same syntax, but the first is preferred.

Syntax

```
LDAP *ldap_init
(
    const char      *hostname,
    int             portno
)
;
```

Parameters

Parameters for initializing an LDAP session is listed in Table 8-3.

Table 8-3   Parameters for Initializing an LDAP Session Functions

Parameter	Description
hostname	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number. The two must be separated by a colon. The hosts are tried in the order listed until a successful connection occurs.
	IPv6 addresses must be enclosed in brackets ([]). The following are examples of valid hostname values:
	host1.example.com
	192.168.1.10
	[2002:ae5:4000:1::8c57:352]
	host1.example.com:3060
	192.168.1.10:3060
	[2002:ae5:4000:1::8c57:352]:3060
portno	Contains the TCP port number to connect to. The default LDAP port of 3060 can be obtained by supplying the constant LDAP_PORT. If hostname includes a port number, portno is ignored.

Usage Notes

ldap_init() and ldap_open() both return a session handle. This is a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. These routines return NULL if the session cannot be initialized. If the session cannot be initialized, check the error reporting mechanism for the operating system to see why the call failed.

Note that if you connect to an LDAPv2 server, one of the LDAP bind calls described later SHOULD be completed before other operations can be performed on the session. LDAPv3 does not require that a bind operation be completed before other operations are performed.

The calling program can set various attributes of the session by calling the routines described in the next section.

8.7 LDAP Session Handle Options

The LDAP session handle returned by ldap_init() is a pointer to an opaque data type representing an LDAP session. In RFC 1823 this data type was a structure exposed to the caller, and various fields in the structure could be set to control aspects of the session, such as size and time limits on searches.

In the interest of insulating callers from inevitable changes to this structure, these aspects of the session are now accessed through a pair of accessor functions, described in this section:

• ldap_get_option and ldap_set_option

8.7.1 ldap_get_option and ldap_set_option

`ldap_get_option()` is used to access the current value of various session-wide parameters. `ldap_set_option()` is used to set the value of these parameters. Note that some options are read only and cannot be set; it is an error to call `ldap_set_option()` and attempt to set a read only option.

Note that if automatic referral following is enabled (the default), any connections created during the course of following referrals inherit the options associated with the session that sent the original request that caused the referrals to be returned.

Syntax

```
int ldap_get_option
(

LDAP            *ld,
int             option,
void            *outvalue

)
;

int ldap_set_option
(

LDAP            *ld,
int             option,
const void      *invalue

)
;


#define LDAP_OPT_ON      ((void *)1)
#define LDAP_OPT_OFF     ((void *)0)
```

Parameters

Table 8-4 lists and describes the parameters for LDAP session handle options.

Table 8-4   Parameters for LDAP Session Handle Functions

Parameters	Description
ld	The session handle. If this is NULL, a set of global defaults is accessed. New LDAP session handles created with `ldap_init()` or `ldap_open()` inherit their characteristics from these global defaults.
option	The name of the option being accessed or set. This parameter should be one of the constants listed and described in Table 8-5. The hexadecimal value of the constant is listed in parentheses after the constant.
outvalue	The address of a place to put the value of the option. The actual type of this parameter depends on the setting of the option parameter. For outvalues of type `char **` and `LDAPControl **`, a copy of the data that is associated with the LDAP session `ld` is returned. Callers should dispose of the memory by calling `ldap_memfree()` or `ldap_controls_free()`, depending on the type of data returned.

Table 8-4   (Cont.) Parameters for LDAP Session Handle Functions

Parameters	Description
`invalue`	A pointer to the value the option is to be given. The actual type of this parameter depends on the setting of the option parameter. The data associated with invalue is copied by the API implementation to allow callers of the API to dispose of or otherwise change their copy of the data after a successful call to `ldap_set_option()`. If a value passed for invalue is invalid or cannot be accepted by the implementation, `ldap_set_option()` should return `-1` to indicate an error.

Constants

Table 8-5 lists and describes the constants for LDAP session handle options.

Table 8-5   Constants for LDAP Session Handle Options

Constant	Type for invalue parameter	Type for outvalue parameter	Description
`LDAP_OPT_API_INFO(0x 00)`	Not applicable. Option is read only.	`LDAPAPIInfo*`	Used to retrieve some basic information about the LDAP API implementation at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only and cannot be set.
`ORA_LDAP_OPT_RFRL_CA CHE`	`void*` (LDAP_OPT_ON) `void*` (LDAP_OPT_OFF)	`int *`	This option determines whether referral cache is enabled or not. If this option is set to `LDAP_OPT_ON`, the cache is enabled; otherwise, the cache is disabled.
`ORA_LDAP_OPT_RFRL_CA CHE_SZ`	`int *`	`int *`	This option sets the size of referral cache. The size is maximum size in terms of number of bytes the cache can grow to. It is set to 1MB by default.
`LDAP_OPT_DEREF(0x02)`	`int *`	`int *`	Determines how aliases are handled during search. It should have one of the following values: `LDAP_DEREF_NEVER (0x00)`, `LDAP_DEREF_SEARCHING (0x01)`, `LDAP_DEREF_FINDING (0x02)`, or `LDAP_DEREF_ALWAYS (0x03)`. The `LDAP_DEREF_SEARCHING` value means aliases are dereferenced during the search but not when locating the base object of the search. The `LDAP_DEREF_FINDING` value means aliases are dereferenced when locating the base object but not during the search. The default value for this option is `LDAP_DEREF_NEVER`.
`LDAP_OPT_SIZELIMIT(0 x03)`	`int *`	`int *`	A limit on the number of entries to return from a search. A value of `LDAP_NO_LIMIT (0)` means no limit. The default value for this option is `LDAP_NO_LIMIT`.

Table 8-5   (Cont.) Constants for LDAP Session Handle Options

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_TIMELIMIT(0x04)	int *	int *	A limit on the number of seconds to spend on a search. A value of LDAP_NO_LIMIT (0) means no limit. This value is passed to the server in the search request only; it does not affect how long the C LDAP API implementation itself waits locally for search results. The timeout parameter passed to ldap_search_ext_s() or ldap_result()—both of which are described later in this document—can be used to specify both a local and server side time limit. The default value for this option is LDAP_NO_LIMIT.
LDAP_OPT_REFERRALS(0x08)	void *(LDAP_OPT_ON) void *(LDAP_OPT_OFF)	int *	Determines whether the LDAP library automatically follows referrals returned by LDAP servers or not. It may be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. By default, this option is turned on.
LDAP_OPT_RESTART(0X09)	void * (LDAP_OPT_ON) void * (LDAP_OPT_OFF)	int *	Determines whether LDAP input and output operations are automatically restarted if they stop prematurely. It may be set to either LDAP_OPT_ON or LDAP_OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. This option is useful if an input or output operation can be interrupted prematurely—by a timer going off, for example. By default, this option is turned off.
LDAP_OPT_PROTOCOL_VERSION(0x11)	int *	int *	This option indicates the version of the LDAP protocol used when communicating with the primary LDAP server. The option should be either LDAP_VERSION2 (2) or LDAP_VERSION3 (3). If no version is set, the default is LDAP_VERSION2 (2).
LDAP_OPT_SERVER_CONTROLS(0x12)	LDAPControl**	LDAPControl***	A default list of LDAP server controls to be sent with each request. **See Also:** Working With Controls

Table 8-5   (Cont.) Constants for LDAP Session Handle Options

Constant	Type for invalue parameter	Type for outvalue parameter	Description
`LDAP_OPT_CLIENT_CONT ROLS(0x13)`	`LDAPControl**`	`LDAPControl***`	A default list of client controls that affect the LDAP session. **See Also:** Working With Controls
`LDAP_OPT_API_FEATURE _INFO(0x15)`	Not applicable. Option is read only.	`LDAPAPIFeatureIn fo *`	Used to retrieve version information about LDAP API extended features at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only. It cannot be set.
`LDAP_OPT_HOST_NAME(0 x30)`	`char *`	`char **`	The host name (or list of hosts) for the primary LDAP server. See the definition of the `hostname` parameter for `ldap_init()` to determine the syntax.
`LDAP_OPT_ERROR_NUMBE R(0x31)`	`int *`	`int *`	The code of the most recent LDAP error during this session.
`LDAP_OPT_ERROR_STRIN G(0x32)`	`char *`	-	The message returned with the most recent LDAP error during this session.
`LDAP_OPT_MATCHED_DN( 0x33)`	`char *`	`char **`	The matched DN value returned with the most recent LDAP error during this session.
`ORA_LDAP_OPT_CONNECT _TIMEOUT (0xD2)`	`int *`	`int *`	This option sets the LDAP connection timeout value in seconds. The connection timeout must be set before calling `ldap_open`. Valid values are 0 – 300 seconds. If the value is set to 0, then the timeout defaults to TCP timeout. If this option is not set, then `ldap_open()` call times out in 15 sec if the host is not reachable.

Usage Notes

Both `ldap_get_option()` and `ldap_set_option()` return `0` if successful and `-1` if an error occurs. If `-1` is returned by either function, a specific error code may be retrieved by calling `ldap_get_option()` with an option value of `LDAP_OPT_ERROR_NUMBER`. Note that there is no way to retrieve a more specific error code if a call to `ldap_get_option()` with an option value of `LDAP_OPT_ERROR_NUMBER` fails.

When a call to `ldap_get_option()` succeeds, the API implementation MUST NOT change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls. When a call to `ldap_get_option()` fails, the only session handle change permitted is setting the LDAP error code (as returned by the `LDAP_OPT_ERROR_NUMBER` option).

When a call to `ldap_set_option()` fails, it must not change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls.

Standards track documents that extend this specification and specify new options should use values for option macros that are between 0x1000 and 0x3FFF inclusive. Private and experimental extensions should use values for the option macros that are between 0x4000 and 0x7FFF inclusive. All values less than 0x1000 and greater than 0x7FFF that are not defined in this document are reserved and should not be used. The following macro must be defined by C LDAP API implementations to aid extension implementers:

```
#define LDAP_OPT_PRIVATE_EXTENSION_BASE 0x4000  /* to 0x7FFF inclusive */
```

8.8 Getting Bind Credentials for Chasing Referrals

The `ldap_set_rebind_proc()` function is used to get the bind credentials of a new server while chasing referrals.

The `ldap_set_rebind_proc()` function is used to set the callback function that the library uses to get the bind credentials for connecting to a new server while chasing LDAP referrals. The library uses the callback function only if `LDAP_OPT_REFERRALS` is set using `ldap_set_option()`. If `ldap_set_rebind_proc()` is not called, then the library uses an anonymous bind to connect to the new server while chasing LDAP referrals.

> **See Also:**
>
> LDAP Session Handle Options

8.8.1 ldap_set_rebind_proc()

The `ldap_set_rebind_proc()` function is used to get the bind credentials of a new server while chasing referrals.

The `ldap_set_rebind_proc()` function is used to set the callback function that the library uses to get the bind credentials for connecting to a new server while chasing LDAP referrals. The library uses the callback function only if `LDAP_OPT_REFERRALS` is set using `ldap_set_option()`. If `ldap_set_rebind_proc()` is not called, then the library uses an anonymous bind to connect to the new server while chasing LDAP referrals.

> **See Also:**
>
> LDAP Session Handle Options

Syntax

```
void ldap_set_rebind_proc
(
LDAP            *ld,
int(*rebindproc) (LDAP   *ld,
                  char  **dnp,
```

```
                char  **passwdp,
                int    *authmethodp,
                int     freeit)
)
```

Parameters

The `reprocbind`() function is the function to be called to get the bind credentials of the new server.

Table 8-6   Parameters for Callback Function and for Setting Callback Function

Parameter	Description
`ld`	The session handle
`rebindproc`	The callback function to be used to get the bind credentials for connecting to a new server while chasing LDAP referrals
`ld`	The session handle to the new server
`dnp`	Pointer to bind dn for new server
passwdp	Pointer to bind password for new server
`authmethodp`	Pointer to authentication method for new server
`freeit`	`0` - returns bind dn pointer, bind password pointer, and bind authentication method pointer for new server
	`1` - frees any memory allocated in previous call

Usage Notes

When the `freeit` parameter value is `0`, then `rebindproc` must return bind dn pointer, bind password pointer, and bind authentication method pointer. When the `freeit` parameter value is 1, then `rebindproc` must free any memory allocated in the previous call. The LDAP library call this function twice, first to get the bind credentials and second time to free the memory.

> ✎ **See Also:**
>
> Setting and Using a Callback Function to Get Credentials When Chasing Referrals.

8.9 Authenticating to the Directory

You can authenticate an LDAP client to an LDAP directory server using LDAP bind functions.

The functions in this section are used to authenticate an LDAP client to an LDAP directory server is described in the following sections:

8.9.1 ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, and ldap_simple_bind_s

The `ldap_sasl_bind()` and `ldap_sasl_bind_s()` functions can be used to do general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer.

The routines both take the DN to bind as, the method to use, as a dotted-string representation of an object identifier (OID) identifying the method, and a `struct berval` holding the credentials. The special constant value `LDAP_SASL_SIMPLE (NULL)` can be passed to request simple authentication, or the simplified routines `ldap_simple_bind()` or `ldap_simple_bind_s()` can be used.

Syntax

```
int ldap_sasl_bind
(

LDAP                    *ld,
const char              *dn,
const char              *mechanism,
const struct berval     *cred,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp
);


int ldap_sasl_bind_s(

LDAP                    *ld,
const char              *dn,
const char              *mechanism,
const struct berval     *cred,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,

);



int ldap_simple_bind(

LDAP                    *ld,
const char              *dn,
const char              *passwd

);



int ldap_simple_bind_s(

LDAP                    *ld,
const char              *dn,
const char              *passwd

);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

- `int ldap_bind( LDAP *ld, const char *dn, const char *cred, int method );`

- `int ldap_bind_s( LDAP *ld, const char *dn, const char *cred, int method );`

- `int ldap_kerberos_bind( LDAP *ld, const char *dn );`

- `int ldap_kerberos_bind_s( LDAP *ld, const char *dn );`

Parameters

Table 8-7 lists and describes the parameters for authenticating to the directory.

Table 8-7   Parameters for Authenticating Directory Functions

Parameter	Description
`ld`	The session handle
`dn`	The name of the entry to bind as
`mechanism`	Either `LDAP_SASL_SIMPLE (NULL)` to get simple authentication, or a text string identifying the SASL method
`cred`	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter.
`passwd`	For `ldap_simple_bind()`, the password to compare to the entry's userPassword attribute
`serverctrls`	List of LDAP server controls
`clientctrls`	List of client controls
`msgidp`	This result parameter is set to the message id of the request if the `ldap_sasl_bind()` call succeeds

Usage Notes

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

The `ldap_sasl_bind()` function initiates an asynchronous bind operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_sasl_bind()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the bind.

The `ldap_simple_bind()` function initiates a simple asynchronous bind operation and returns the message id of the operation initiated. A subsequent call to `ldap_result()`, described in, can be used to obtain the result of the bind. In case of error, `ldap_simple_bind()` returns `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_sasl_bind_s()` and `ldap_simple_bind_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

Note that if an LDAPv2 server is contacted, no other operations over the connection can be attempted before a bind call has successfully completed.

Subsequent bind calls can be used to re-authenticate over the same connection, and multistep SASL sequences can be accomplished through a sequence of calls to `ldap_sasl_bind()` or `ldap_sasl_bind_s()`.

> ✎ **See Also:**
>
> Handling Errors and Parsing Results for more information about possible errors and how to interpret them.

8.10 SASL Authentication Using Oracle Extensions

SASL Authentication is done by using `ora_ldap_init_SASL` function.

This section contains the following topics:

- ora_ldap_init_SASL
- ora_ldap_create_cred_hdl, ora_ldap_set_cred_props, ora_ldap_get_cred_props, and ora_ldap_free_cred_hdl

8.10.1 ora_ldap_init_SASL

The function `ora_ldap_init_SASL()` can be used for SASL based authentication. It performs authentication based on the mechanism specified as one of its input arguments.

This function encapsulates the SASL handshake between the client and the directory server for various standard SASL mechanisms thereby reducing the coding effort involved in establishing a SASL-based connection to the directory server.

Syntax

```
int ora_ldap_init_SASL
(
        OraLdapClientCtx * clientCtx,
        LDAP            *ld,
        char            * dn,
        char            * mechanism,
        OraLdapHandle       cred,
        LDAPControl            **serverctrls,
        LDAPControl            **clientctrls
);
```

Parameters

The parameters are listed in the following table.

Table 8-8   Parameters for ora_ldap_init_sasl() Function

Parameter	Description
clientCtx	C API Client context. This can be managed using `ora_ldap_init_clientctx()` and `ora_ldap_free_clientctx()` functions.
ld	Ldap session handle.
dn	User DN to be authenticated.
mechanism	SASL mechanism.
cred	Credentials needed for SASL authentication.
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Usage Notes

The cred parameter is a SASL credential handle for the user. This handle can be managed using `ora_ldap_create_cred_hdl()`, `ora_ldap_set_cred_props()` and `ora_ldap_free_cred_hdl()` functions.

Supported SASL mechanisms:

- DIGEST-MD5

  The Oracle Internet Directory SASL API supports the authentication-only mode of DIGEST-MD5. The other two authentication modes addressing data privacy and data integrity are yet to be supported.

  While authenticating against Oracle Internet Directory, the DN of the user has to be normalized before it is sent across to the server. This can be done either outside the SASL API using the `ora_ldap_normalize_dn()` function before the DN is passed on to the SASL API or with the SASL API by setting the `ORA_LDAP_CRED_SASL_NORM_AUTHDN` option in SASL credentials handle using `ora_ldap_set_cred_handle()`.

- EXTERNAL:

  The SASL API and SASL implementation in Oracle Internet Directory use SSL authentication as one of the external authentication mechanisms.

  Using this mechanism requires that the SSL connection (mutual authentication mode) be established to the directory server by using the `ora_ldap_init_SSL()` function. The `ora_ldap_init_SASL()` function can then be invoked with the `mechanism` argument as `EXTERNAL`. The directory server would then authenticate the user based on the user credentials in SSL connection.

8.10.2 ora_ldap_create_cred_hdl, ora_ldap_set_cred_props, ora_ldap_get_cred_props, and ora_ldap_free_cred_hdl

Use these functions to create and manage SASL credential handles. The `ora_ldap_create_cred_hdl` function should be used to create a SASL credential handle of certain type based on the type of mechanism used for SASL authentication. The `ora_ldap_set_cred_props()` function can be used to add relevant credentials to the handle needed for SASL authentication.

The `ora_ldap_get_cred_props()` function can be used for retrieving the properties stored in the credential handle, and the `ora_ldap_free_cred_hdl()` function should be used to destroy the handle after its use.

Syntax

```
OraLdapHandle ora_ldap_create_cred_hdl
(
        OraLdapClientCtx * clientCtx,
        int                 credType
);

OraLdapHandle ora_ldap_set_cred_props
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle        cred,
        int                  String[],
        void             *   inProperty
);
OraLdapHandle ora_ldap_get_cred_props
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle    cred,
        int                  String[],
        void             *   outProperty
);

OraLdapHandle ora_ldap_free_cred_hdl
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle    cred
);
```

Parameters

The parameters are listed in the following table.

Table 8-9   Parameters for Managing SASL Credential Functions

Parameter	Description
clientCtx	C API Client context. This can be managed using `ora_ldap_init_clientctx()` and `ora_ldap_free_clientctx()` functions.
credType	Type of credential handle specific to SASL mechanism.

Table 8-9   (Cont.) Parameters for Managing SASL Credential Functions

Parameter	Description
`cred`	Credential handle containing SASL credentials needed for a specific SASL mechanism for SASL authentication.
`String[]`	Type of credential, which must be added to credential handle.
`inProperty`	One of the SASL Credentials to be stored in credential handle.
`outProperty`	One of the SASL credentials stored in credential handle.

Usage Notes

None

8.11 Working With Controls

LDAPv3 operations can be extended through the use of controls. Controls can be sent to a server or returned to the client with any LDAP message. These controls are referred to as server controls.

The LDAP API also supports a client-side extension mechanism through the use of client controls. These controls affect the behavior of the LDAP API only and are never sent to a server.

A common data structure is used to represent both types of controls.

8.11.1 ldapcontrol

The `ldapcontrol` data structure has control type, data associated with the control and an element which indicates whether the control is critical or not.

Syntax

```
typedef struct ldapcontrol
{

char           *ldctl_oid;
struct berval   ldctl_value;
char            ldctl_iscritical;

} LDAPControl;
```

Parameters

The fields in the `ldapcontrol` structure are described in Table 8-10.

Table 8-10   Fields in ldapcontrol Structure

Field	Description
`ldctl_oid`	The control type, represented as a string.
`ldctl_value`	The data associated with the control (if any). To specify a zero-length value, set `ldctl_value.bv_len` to zero and `ldctl_value.bv_val` to a zero-length string. To indicate that no data is associated with the control, set `ldctl_value.bv_val` to `NULL`.
`ldctl_iscritical`	Indicates whether the control is critical of not. If this field is nonzero, the operation is only carried out if the control is recognized by the server or the client. Note that the LDAP unbind and abandon operations have no server response. Clients should not mark server controls critical when used with these two operations.

> **See Also:**
>
> Extensions to the LDAP Protocol for more information about controls.

Usage Notes

Some LDAP API calls allocate an `ldapcontrol` structure or a `NULL`-terminated array of `ldapcontrol` structures. The following routines can be used to dispose of a single control or an array of controls:

```
void ldap_control_free( LDAPControl *ctrl );
void ldap_controls_free( LDAPControl **ctrls );
```

If the `ctrl` or `ctrls` parameter is `NULL`, these calls do nothing.

A set of controls that affect the entire session can be set using the `ldap_set_option()` function described in ldap_get_option and ldap_set_option A list of controls can also be passed directly to some LDAP API calls such as `ldap_search_ext()`, in which case any controls set for the session through the use of `ldap_set_option()` are ignored. Control lists are represented as a `NULL`-terminated array of pointers to `ldapcontrol` structures.

Server controls are defined by LDAPv3 protocol extension documents; for example, a control has been proposed to support server-side sorting of search results.

One client control is defined in this chapter (described in the following section).

Client-Controlled Referral Processing

As described previously in Initializing an LDAP Session and Usage Notes applications can enable and disable automatic chasing of referrals on a session-wide basic by using the `ldap_set_option()` function with the `LDAP_OPT_REFERRALS` option. It is also useful to govern automatic referral chasing on per-request basis. A client control with an object identifier (OID) of `1.2.840.113556.1.4.616` exists to provide this functionality.

```
/* OID for referrals client control */
#define LDAP_CONTROL_REFERRALS            "1.2.840.113556.1.4.616"

/* Flags for referrals client control value */
```

```
#define LDAP_CHASE_SUBORDINATE_REFERRALS      0x00000020U
#define LDAP_CHASE_EXTERNAL_REFERRALS         0x00000040U
```

To create a referrals client control, the `ldctl_oid` field of an `LDAPControl` structure must be set to `LDAP_CONTROL_REFERRALS ("1.2.840.113556.1.4.616")` and the `ldctl_value` field must be set to a four-octet value that contains a set of flags. The `ldctl_value.bv_len` field must always be set to `4`. The `ldctl_value.bv_val` field must point to a four-octet integer flags value. This flags value can be set to zero to disable automatic chasing of referrals and LDAPv3 references altogether. Alternatively, the flags value can be set to the value `LDAP_CHASE_SUBORDINATE_REFERRALS (0x00000020U)` to indicate that only LDAPv3 search continuation references are to be automatically chased by the API implementation, to the value `LDAP_CHASE_EXTERNAL_REFERRALS (0x00000040U)` to indicate that only LDAPv3 referrals are to be automatically chased, or the logical OR of the two flag values `(0x00000060U)` to indicate that both referrals and references are to be automatically chased.

> ✎ **See Also:**
>
> Directory Schema Administration in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for more information about object identifiers.

8.12 Closing the Session

You can use LDAP functions to unbind, close and open connections.

The following functions are used to unbind from the directory, to close open connections, and to dispose of the session handle.

- ldap_unbind
- ldap_unbind_ext
- ldap_unbind_s

8.12.1 ldap_unbind_ext(), ldap_unbind(), and ldap_unbind_s()

`ldap_unbind_ext()`, `ldap_unbind()`, and `ldap_unbind_s()` all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning. Note, however, that there is no server response to an LDAP unbind operation.

All three of the unbind functions return `LDAP_SUCCESS` (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle `ld` is invalid and it is illegal to make any further LDAP API calls using `ld`.

The `ldap_unbind()` and `ldap_unbind_s()` functions behave identically. The `ldap_unbind_ext()` function allows server and client controls to be included explicitly, but note that since there is no server response to an unbind request there is no way to receive a response to a server control sent with an unbind request.

Syntax

```
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
LDAPControl **clientctrls );
int ldap_unbind( LDAP *ld );
int ldap_unbind_s( LDAP *ld );
```

Parameters

The parameters are listed in the following table.

Table 8-11   Parameters for Closing the Session Functions

Parameter	Description
ld	The session handle
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Usage Notes

None

8.13 Performing LDAP Operations

LDAP operations are used perfrom search the LDAP directory, compare, and modify the directory.

Use the functions in this section to search the LDAP directory and to return a requested set of attributes for each entry matched:

- ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s
- ldap_compare_ext, ldap_compare_ext_s, ldap_compare, and ldap_compare_s
- ldap_modify_ext, ldap_modify_ext_s, ldap_modify, and ldap_modify_s
- ldap_rename and ldap_rename_s
- ldap_add_ext, ldap_add_ext_s, ldap_add, and ldap_add_s
- ldap_delete_ext, ldap_delete_ext_s, ldap_delete, and ldap_delete_s
- ldap_extended_operation and ldap_extended_operation_s
-

8.13.1 ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s

The `ldap_search_ext()` function initiates an asynchronous search operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_search_ext()` places the message id of the request in `*msgidp`. A

subsequent call to `ldap_result()` can be used to obtain the results from the search. These results can be parsed using the result parsing routines described in detail later.

Similar to `ldap_search_ext()`, the `ldap_search()` function initiates an asynchronous search operation and returns the message id of the operation initiated. As for `ldap_search_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the bind. In case of error, `ldap_search()` returns `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_search_ext_s()`, `ldap_search_s()`, and `ldap_search_st()` functions all return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. Entries returned from the search, if any, are contained in the `res` parameter. This parameter is opaque to the caller. Entries, attributes, values, and so on, can be extracted by calling the parsing routines described in this section. The results contained in `res` should be freed when no longer in use by calling `ldap_msgfree()`, which is described later.

The `ldap_search_ext()` and `ldap_search_ext_s()` functions support LDAPv3 server controls, client controls, and allow varying size and time limits to be easily specified for each search operation. The `ldap_search_st()` function is identical to `ldap_search_s()` except that it takes an additional parameter specifying a local timeout for the search. The local search timeout is used to limit the amount of time the API implementation waits for a search to complete. After the local search timeout expires, the API implementation sends an abandon operation to stop the search operation.

> ✎ **See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_search_ext
(

LDAP           *ld,
const char     *base,
int            scope,
const char     *filter,
char           **attrs,
int            attrsonly,
LDAPControl    **serverctrls,
LDAPControl    **clientctrls,
struct timeval *timeout,
int            sizelimit,
int            *msgidp

);

int ldap_search_ext_s
(

LDAP           *ld,
const char     *base,
```

```
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
struct timeval  *timeout,
int             sizelimit,
LDAPMessage     **res

);

int ldap_search
(

LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly

);

int ldap_search_s
(

LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
LDAPMessage     **res

);

int ldap_search_st
);

LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
struct timeval  *timeout,
LDAPMessage     **res

);
```

Parameters

Table 8-12 lists and describes the parameters for search operations.

Table 8-12   Parameters for Search Operation Functions

Parameter	Description
ld	The session handle.
base	The DN of the entry at which to start the search.
scope	One of `LDAP_SCOPE_BASE (0x00)`, `LDAP_SCOPE_ONELEVEL (0x01)`, or `LDAP_SCOPE_SUBTREE (0x02)`, indicating the scope of the search.
filter	A character string representing the search filter. The value `NULL` can be passed to indicate that the filter "`(objectclass=*)`" which matches all entries is to be used. Note that if the caller of the API is using LDAPv2, only a subset of the filter functionality can be successfully used.
attrs	A `NULL`-terminated array of strings indicating which attributes to return for each matching entry. Passing `NULL` for this parameter causes all available user attributes to be retrieved. The special constant string `LDAP_NO_ATTRS` ("`1.1`") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string `LDAP_ALL_USER_ATTRS` ("`*`") can be used in the `attrs` array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
timeout	For the `ldap_search_st()` function, this specifies the local search timeout value (if it is `NULL`, the timeout is infinite). If a zero timeout (where `tv_sec` and `tv_usec` are both zero) is passed, API implementations should return `LDAP_PARAM_ERROR`. For the `ldap_search_ext()` and `ldap_search_ext_s()` functions, the timeout parameter specifies both the local search timeout value and the operation time limit that is sent to the server within the search request. Passing a `NULL` value for timeout causes the global default timeout stored in the LDAP session handle (set by using `ldap_set_option()` with the `LDAP_OPT_TIMELIMIT` parameter) to be sent to the server with the request but an infinite local search timeout to be used. If a zero timeout (where `tv_sec` and `tv_usec` are both zero) is passed in, API implementations should return `LDAP_PARAM_ERROR`. If a zero value for `tv_sec` is used but `tv_usec` is nonzero, an operation time limit of 1 should be passed to the LDAP server as the operation time limit. For other values of `tv_sec`, the `tv_sec` value itself should be passed to the LDAP server.
sizelimit	For the `ldap_search_ext()` and `ldap_search_ext_s()` calls, this is a limit on the number of entries to return from the search. A value of `LDAP_NO_LIMIT (0)` means no limit.
res	For the synchronous calls, this is a result parameter which contains the results of the search upon completion of the call. If no results are returned, `*res` is set to `NULL`.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Table 8-12   (Cont.) Parameters for Search Operation Functions

Parameter	Description
`msgidp`	This result parameter is set to the message id of the request if the `ldap_search_ext()` call succeeds.There are three options in the session handle `ld` which potentially affect how the search is performed. They are: • `LDAP_OPT_SIZELIMIT`—A limit on the number of entries to return from the search. A value of `LDAP_NO_LIMIT (0)` means no limit. Note that the value from the session handle is ignored when using the `ldap_search_ext()` or `ldap_search_ext_s()` functions. • `LDAP_OPT_TIMELIMIT`—A limit on the number of seconds to spend on the search. A value of `LDAP_NO_LIMIT (0)` means no limit. Note that the value from the session handle is ignored when using the `ldap_search_ext()` or `ldap_search_ext_s()` functions. • `LDAP_OPT_DEREF`—One of `LDAP_DEREF_NEVER (0x00)`, `LDAP_DEREF_SEARCHING (0x01)`, `LDAP_DEREF_FINDING (0x02)`, or `LDAP_DEREF_ALWAYS (0x03)`, specifying how aliases are handled during the search. The `LDAP_DEREF_SEARCHING` value means aliases are dereferenced during the search but not when locating the base object of the search. The `LDAP_DEREF_FINDING` value means aliases are dereferenced when locating the base object but not during the search.

Usage Notes

Reading an Entry

LDAP does not support a read operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to read, scope set to `LDAP_SCOPE_BASE`, and filter set to `"(objectclass=*)"` or `NULL`. The `attrs` parameter contains the list of attributes to return.

Listing the Children of an Entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to list, scope set to LDAP_SCOPE_ONELEVEL, and filter set to "(objectclass=*)" or NULL. The parameter attrs contains the list of attributes to return for each child entry.

8.13.2 ldap_compare_ext, ldap_compare_ext_s, ldap_compare, and ldap_compare_s

The `ldap_compare_ext()` function initiates an asynchronous compare operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_compare_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the compare.

Use these routines to compare an attribute value assertion against an LDAP entry.

Similar to `ldap_compare_ext()`, the `ldap_compare()` function initiates an asynchronous compare operation and returns the message id of the operation initiated. As for `ldap_compare_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the bind. In case of error, `ldap_compare()` returns -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_compare_ext_s()` and `ldap_compare_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_compare_ext()` and `ldap_compare_ext_s()` functions support LDAPv3 server controls and client controls.

> **✎ See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_compare_ext
(

LDAP                    *ld,
const char              *dn,
const char              *attr,
const struct berval     *bvalue,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp

);

int ldap_compare_ext_s
(

LDAP                    *ld,
const char              *dn,
const char              *attr,
const struct berval     *bvalue,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls

);

int ldap_compare
(

LDAP                    *ld,
const char              *dn,
const char              *attr,
const char              *value

);
int ldap_compare_s
(

LDAP                    *ld,
const char              *dn,
const char              *attr,
const char              *value

);
```

Parameters

Table 8-13 lists and describes the parameters for compare operations.

Table 8-13   Parameters for Compare Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
bvalue	The attribute value to compare against those found in the given entry. This parameter is used in the extended routines and is a pointer to a `struct berval` so it is possible to compare binary values.
value	A string attribute value to compare against, used by the `ldap_compare()` and `ldap_compare_s()` functions. Use `ldap_compare_ext()` or `ldap_compare_ext_s()` if you need to compare binary values.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter is set to the message id of the request if the `ldap_compare_ext()` call succeeds.

Usage Notes

None

8.13.3 ldap_modify_ext, ldap_modify_ext_s, ldap_modify, and ldap_modify_s

Use these routines to modify an existing LDAP entry. The `ldap_modify_ext()` function initiates an asynchronous modify operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_modify_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the modify.

Similar to `ldap_modify_ext()`, the `ldap_modify()` function initiates an asynchronous modify operation and returns the message id of the operation initiated. As for `ldap_modify_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the modify. In case of error, `ldap_modify()` returns -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_modify_ext_s()` and `ldap_modify_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_modify_ext()` and `ldap_modify_ext_s()` functions support LDAPv3 server controls and client controls.

> **See Also:**
>
> Handling Errors and Parsing Results for more information about possible errors and how to interpret them.

Syntax

```
typedef struct ldapmod
{

int            mod_op;
char           *mod_type;
union mod_vals_u
  {

  char           **modv_strvals;
  struct berval  **modv_bvals;
  } mod_vals;

} LDAPMod;

#define mod_values      mod_vals.modv_strvals
#define mod_bvalues     mod_vals.modv_bvals


int ldap_modify_ext
(

LDAP           *ld,
const char     *dn,
LDAPMod        **mods,
LDAPControl    **serverctrls,
LDAPControl    **clientctrls,
int            *msgidp

);

int ldap_modify_ext_s
(

LDAP           *ld,
const char     *dn,
LDAPMod        **mods,
LDAPControl    **serverctrls,
LDAPControl    **clientctrls

);

int ldap_modify
(

LDAP           *ld,
const char     *dn,
```

```
LDAPMod          **mods
);


int ldap_modify_s
(

LDAP           *ld,
const char     *dn,
LDAPMod        **mods

);
```

Parameters

Table 8-14 lists and describes the parameters for modify operations.

Table 8-14   Parameters for Modify Operation Function

Parameter	Description
ld	The session handle
dn	The name of the entry to modify
mods	A NULL-terminated array of modifications to make to the entry
serverctrls	List of LDAP server controls
clientctrls	List of client controls
msgidp	This result parameter is set to the message id of the request if the ldap_modify_ext() call succeeds

Table 8-15 lists and describes the fields in the LDAPMod structure.

Table 8-15   Fields in LDAPMod Structure

Field	Description
mod_op	The modification operation to perform. It must be one of LDAP_MOD_ADD (0x00), LDAP_MOD_DELETE (0x01), or LDAP_MOD_REPLACE (0x02). This field also indicates the type of values included in the mod_vals union. It is logically ORed with LDAP_MOD_BVALUES (0x80) to select the mod_bvalues form. Otherwise, the mod_values form is used.
mod_type	The type of the attribute to modify.
mod_vals	The values (if any) to add, delete, or replace. Only one of the mod_values or mod_bvalues variants can be used, selected by ORing the mod_op field with the constant LDAP_MOD_BVALUES. mod_values is a NULL-terminated array of zero-terminated strings and mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

Usage Notes

For `LDAP_MOD_ADD` modifications, the given values are added to the entry, creating the attribute if necessary.

For `LDAP_MOD_DELETE` modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the mod_vals field can be set to `NULL`.

For `LDAP_MOD_REPLACE` modifications, the attribute has the listed values after the modification, having been created if necessary, or removed if the `mod_vals` field is `NULL`. All modifications are performed in the order in which they are listed.

8.13.4 ldap_rename and ldap_rename_s

Use these routines to change the name of an entry. The `ldap_rename()` function initiates an asynchronous modify DN operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_rename()` places the DN message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the rename.

The synchronous `ldap_rename_s()` returns the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_rename()` and `ldap_rename_s()` functions both support LDAPv3 server controls and client controls.

> **See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_rename
(

LDAP           *ld,
const char     *dn,
const char     *newrdn,
const char     *newparent,
int            deleteoldrdn,
LDAPControl    **serverctrls,
LDAPControl    **clientctrls,
int            *msgidp

);

int ldap_rename_s
(

LDAP           *ld,
const char     *dn,
```

```
const char      *newrdn,
const char      *newparent,
int             deleteoldrdn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls

);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

```
int ldap_modrdn
(

LDAP            *ld,
const char      *dn,
const char      *newrdn

);

int ldap_modrdn_s
(

LDAP            *ld,
const char      *dn,
const char      *newrdn

);

int ldap_modrdn2
(

LDAP            *ld,
const char      *dn,
const char      *newrdn,
int             deleteoldrdn

);

int ldap_modrdn2_s
(

LDAP            *ld,
const char      *dn,
const char      *newrdn,
int             deleteoldrdn

);
```

Parameters

Table 8-16 lists and describes the parameters for rename operations.

Table 8-16   Parameters for Rename Operation Function

Parameter	Description
ld	The session handle.

Table 8-16   (Cont.) Parameters for Rename Operation Function

Parameter	Description
dn	The name of the entry whose DN is to be changed.
newrdn	The new RDN to give the entry.
newparent	The new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is changed. The root DN should be specified by passing a zero length string, "". The newparent parameter should always be NULL when using version 2 of the LDAP protocol; otherwise the server's behavior is undefined.
deleteoldrdn	This parameter only has meaning on the rename routines if newrdn is different than the old RDN. It is a boolean value, if nonzero indicating that the old RDN value is to be removed, if zero indicating that the old RDN value is to be retained as non-distinguished values of the entry.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter is set to the message id of the request if the ldap_rename() call succeeds.

Usage Notes

None

8.13.5 ldap_add_ext, ldap_add_ext_s, ldap_add, and ldap_add_s

Use these functions to add entries to the LDAP directory. The ldap_add_ext() function initiates an asynchronous add operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_add_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the add.

Similar to ldap_add_ext(), the ldap_add() function initiates an asynchronous add operation and returns the message id of the operation initiated. As for ldap_add_ext(), a subsequent call to ldap_result() can be used to obtain the result of the add. In case of error, ldap_add() returns -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_add_ext_s() and ldap_add_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The ldap_add_ext() and ldap_add_ext_s() functions support LDAPv3 server controls and client controls.

> **✎ See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors
> and how to interpret them.

Syntax

```
int ldap_add_ext
(

LDAP            *ld,
const char      *dn,
LDAPMod         **attrs,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
int             *msgidp

);

int ldap_add_ext_s
(

LDAP            *ld,
const char      *dn,
LDAPMod         **attrs,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls

);

int ldap_add
(

LDAP            *ld,
const char      *dn,
LDAPMod         **attrs

);

int ldap_add_s
(

LDAP            *ld,
const char      *dn,
LDAPMod         **attrs

);
```

Parameters

Table 8-17 lists and describes the parameters for add operations.

Table 8-17   Parameters for Add Operation Function

Parameter	Description
ld	The session handle.
dn	The name of the entry to add.
attrs	The entry attributes, specified using the LDAPMod structure defined for ldap_modify(). The mod_type and mod_vals fields must be filled in. The mod_op field is ignored unless ORed with the constant LDAP_MOD_BVALUES, used to select the mod_bvalues case of the mod_vals union.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter is set to the message id of the request if the ldap_add_ext() call succeeds.

Usage Notes

Note that the parent of the entry being added must already exist or the parent must be empty—that is, equal to the root DN—for an add to succeed.

8.13.6 ldap_delete_ext, ldap_delete_ext_s, ldap_delete, and ldap_delete_s

Use these functions to delete a leaf entry from the LDAP directory. The ldap_delete_ext() function initiates an asynchronous delete operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_delete_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the delete.

Similar to ldap_delete_ext(), the ldap_delete() function initiates an asynchronous delete operation and returns the message id of the operation initiated. As for ldap_delete_ext(), a subsequent call to ldap_result() can be used to obtain the result of the delete. In case of error, ldap_delete() returns -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_delete_ext_s() and ldap_delete_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The ldap_delete_ext() and ldap_delete_ext_s() functions support LDAPv3 server controls and client controls.

> **✎ See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors
> and how to interpret them.

Syntax

```
int ldap_delete_ext
(

LDAP            *ld,
const char      *dn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
int             *msgidp

);

int ldap_delete_ext_s
(
LDAP            *ld,

const char      *dn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls

);
```

int ldap_delete

```
(

LDAP            *ld,
const char      *dn

);

int ldap_delete_s
(

LDAP            *ld,
const char      *dn

);
```

Parameters

Table 8-18 lists and describes the parameters for delete operations.

Table 8-18   Parameters for Delete Operation Function

Parameter	Description
ld	The session handle.

Table 8-18   (Cont.) Parameters for Delete Operation Function

Parameter	Description
dn	The name of the entry to delete.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter is set to the message id of the request if the `ldap_delete_ext()` call succeeds.

Usage Notes

Note that the entry to delete must be a leaf entry—that is, it must have no children. Deletion of entire subtrees in a single operation is not supported by LDAP.

8.13.7 ldap_extended_operation and ldap_extended_operation_s

These routines enable extended LDAP operations to be passed to the server, providing a general protocol extensibility mechanism.

The `ldap_extended_operation()` function initiates an asynchronous extended operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_extended_operation()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the extended operation which can be passed to `ldap_parse_extended_result()` to obtain the object identifier (OID) and data contained in the response.

The synchronous `ldap_extended_operation_s()` function returns the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. The `retoid` and `retdata` parameters are filled in with the OID and data from the response. If no OID or data was returned, these parameters are set to `NULL`.

The `ldap_extended_operation()` and `ldap_extended_operation_s()` functions both support LDAPv3 server controls and client controls.

> **✏️ See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_extended_operation
(
```

```
LDAP                    *ld,
const char              *requestoid,
const struct berval     *requestdata,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp

);

int ldap_extended_operation_s
(

LDAP                    *ld,
const char              *requestoid,
const struct berval     *requestdata,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
char                    **retoidp,
struct berval           **retdatap

);
```

Parameters

Table 8-19 lists and describes the parameters for extended operations.

Table 8-19   Parameters for Extended Operation Function

Parameter	Description
ld	The session handle
requestoid	The dotted-OID text string naming the request
requestdata	The arbitrary data needed by the operation (if NULL, no data is sent to the server)
serverctrls	List of LDAP server controls
clientctrls	List of client controls
msgidp	This result parameter is set to the message id of the request if the ldap_extended_operation() call succeeds.
retoidp	Pointer to a character string that is set to an allocated, dotted-OID text string returned by the server. This string should be disposed of using the ldap_memfree() function. If no OID was returned, *retoidp is set to NULL.
retdatap	Pointer to a berval structure pointer that is set an allocated copy of the data returned by the server. This struct berval should be disposed of using ber_bvfree(). If no data is returned, *retdatap is set to NULL.

Usage Notes

None

8.14 Abandoning an Operation

`ldap_abandon_ext()` abandons the operation with message id `msgid` and returns the constant `LDAP_SUCCESS` if the abandon was successful or another LDAP error code if not.

`ldap_abandon()` is identical to `ldap_abandon_ext()` except that it does not accept client or server controls and it returns zero if the abandon was successful, `-1` otherwise.

After a successful call to `ldap_abandon()` or `ldap_abandon_ext()`, results with the given message id are never returned from a subsequent call to `ldap_result()`. There is no server response to LDAP abandon operations.

8.14.1 ldap_abandon_ext()

`ldap_abandon_ext()` abandons the operation with message id msgid and returns the constant LDAP_SUCCESS if the abandon was successful or another LDAP error code if not.

Syntax

```
int ldap_abandon_ext
(

LDAP            *ld,
int             msgid,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls

);

int ldap_abandon
(

LDAP            *ld,
int             msgid

);
```

Parameters

Table 8-20 lists and describes the parameters for abandoning an operation.

Table 8-20   Parameters for Abandoning an Operation Function

Parameter	Description
ld	The session handle.
msgid	The message id of the request to be abandoned.
serverctrls	List of LDAP server controls.

Table 8-20   (Cont.) Parameters for Abandoning an Operation Function

Parameter	Description
clientctrls	List of client controls.

> ✎ **See Also:**
>
> "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Usage Notes

None

8.15 Obtaining Results and Peeking Inside LDAP Messages

The result of an operation initiated asynchronously is returned by `ldap_result()` function.

Use the functions in this section to return the result of an operation initiated asynchronously. They identify messages by type and by ID.

8.15.1 ldap_result, ldap_msgtype, and ldap_msgid

`ldap_result()` is used to obtain the result of a previous asynchronously initiated operation. Note that depending on how it is called, `ldap_result()` can actually return a list or "chain" of result messages.

The `ldap_result()` function only returns messages for a single request, so for all LDAP operations other than search only one result message is expected; that is, the only time the "result chain" can contain more than one message is if results from a search operation are returned.

After a chain of messages has been returned to the caller, it is no longer tied in any caller-visible way to the LDAP request that produced it. Therefore, a chain of messages returned by calling `ldap_result()` or by calling a synchronous search routine is never affected by subsequent LDAP API calls (except for `ldap_msgfree()` which is used to dispose of a chain of messages).

`ldap_msgfree()` frees the result messages (possibly an entire chain of messages) obtained from a previous call to `ldap_result()` or from a call to a synchronous search routine.

`ldap_msgtype()` returns the type of an LDAP message. `ldap_msgid()` returns the message ID of an LDAP message.

Syntax

```
int ldap_result
(
```

```
LDAP            *ld,
int             msgid,
int             all,
struct timeval  *timeout,
LDAPMessage     **res

);
int ldap_msgfree( LDAPMessage *res );
int ldap_msgtype( LDAPMessage *res );
int ldap_msgid( LDAPMessage *res );
```

Parameters

Table 8-21 lists and describes the parameters for obtaining results and peeling inside LDAP messages.

Table 8-21   Parameters for Obtaining Results and Peeking Inside LDAP Messages Function

Parameter	Description
ld	The session handle.
msgid	The message id of the operation whose results are to be returned, the constant LDAP_RES_UNSOLICITED (0) if an unsolicited result is desired, or the constant LDAP_RES_ANY (-1) if any result is desired.
all	Specifies how many messages is retrieved in a single call to ldap_result(). This parameter only has meaning for search results. Pass the constant LDAP_MSG_ONE (0x00) to retrieve one message at a time. Pass LDAP_MSG_ALL (0x01) to request that all results of a search be received before returning all results in a single chain. Pass LDAP_MSG_RECEIVED (0x02) to indicate that all messages retrieved so far are to be returned in the result chain.
timeout	A timeout specifying how long to wait for results to be returned. A NULL value causes ldap_result() to block until results are available. A timeout value of zero seconds specifies a polling behavior.
res	For ldap_result(), a result parameter that contains the result of the operation. If no results are returned, *res is set to NULL. For ldap_msgfree(), the result chain to be freed, obtained from a previous call to ldap_result(), ldap_search_s(), or ldap_search_st(). If res is NULL, nothing is done and ldap_msgfree() returns zero.

Usage Notes

Upon successful completion, ldap_result() returns the type of the first result returned in the res parameter. This is one of the following constants.

LDAP_RES_BIND (0x61)

LDAP_RES_SEARCH_ENTRY (0x64)

LDAP_RES_SEARCH_REFERENCE (0x73) -- new in LDAPv3

LDAP_RES_SEARCH_RESULT (0x65)

LDAP_RES_MODIFY (0x67)

```
LDAP_RES_ADD (0x69)

LDAP_RES_DELETE (0x6B)

LDAP_RES_MODDN (0x6D)

LDAP_RES_COMPARE (0x6F)

LDAP_RES_EXTENDED (0x78)
```
`LDAP_RES_EXTENDED (0x78)` -- new in LDAPv3

`ldap_result()` returns `0` if the timeout expired and `-1` if an error occurs, in which case the error parameters of the LDAP session handle is set accordingly.

`ldap_msgfree()` frees each message in the result chain pointed to by `res` and returns the type of the last message in the chain. If `res` is `NULL`, then nothing is done and the value zero is returned.

`ldap_msgtype()` returns the type of the LDAP message it is passed as a parameter. The type is one of the types listed previously, or `-1` on error.

`ldap_msgid()` returns the message ID associated with the LDAP message passed as a parameter, or `-1` on error.

8.16 Obtaining Results and Peeking Inside LDAP Messages

Use the functions in this section to return the result of an operation initiated asynchronously. They identify messages by type and by ID.

8.16.1 ldap_result, ldap_msgtype, and ldap_msgid

`ldap_result()` is used to obtain the result of a previous asynchronously initiated operation. Note that depending on how it is called, `ldap_result()` can actually return a list or "chain" of result messages. The `ldap_result()` function only returns messages for a single request, so for all LDAP operations other than search only one result message is expected; that is, the only time the "result chain" can contain more than one message is if results from a search operation are returned.

After a chain of messages has been returned to the caller, it is no longer tied in any caller-visible way to the LDAP request that produced it. Therefore, a chain of messages returned by calling `ldap_result()` or by calling a synchronous search routine is never affected by subsequent LDAP API calls (except for `ldap_msgfree()` which is used to dispose of a chain of messages).

`ldap_msgfree()` frees the result messages (possibly an entire chain of messages) obtained from a previous call to `ldap_result()` or from a call to a synchronous search routine.

`ldap_msgtype()` returns the type of an LDAP message. `ldap_msgid()` returns the message ID of an LDAP message.

Syntax

```
int ldap_result
(

LDAP            *ld,
int             msgid,
int             all,
```

```
struct timeval  *timeout,
LDAPMessage     **res

);
int ldap_msgfree( LDAPMessage *res );
int ldap_msgtype( LDAPMessage *res );
int ldap_msgid( LDAPMessage *res );
```

Parameters

Table 8-22 lists and describes the parameters for obtaining results and peeking inside LDAP messages.

Table 8-22   Parameters for LDAP Messages Functions

Parameter	Description
ld	The session handle.
msgid	The message id of the operation whose results are to be returned, the constant `LDAP_RES_UNSOLICITED (0)` if an unsolicited result is desired, or the constant `LDAP_RES_ANY (-1)` if any result is desired.
all	Specifies how many messages is retrieved in a single call to `ldap_result()`. This parameter only has meaning for search results. Pass the constant `LDAP_MSG_ONE (0x00)` to retrieve one message at a time. Pass `LDAP_MSG_ALL (0x01)` to request that all results of a search be received before returning all results in a single chain. Pass `LDAP_MSG_RECEIVED (0x02)` to indicate that all messages retrieved so far are to be returned in the result chain.
timeout	A timeout specifying how long to wait for results to be returned. A `NULL` value causes `ldap_result()` to block until results are available. A timeout value of zero seconds specifies a polling behavior.
res	For `ldap_result()`, a result parameter that contains the result of the operation. If no results are returned, `*res` is set to `NULL`. For `ldap_msgfree()`, the result chain to be freed, obtained from a previous call to `ldap_result()`, `ldap_search_s()`, or `ldap_search_st()`. If res is `NULL`, nothing is done and `ldap_msgfree()` returns zero.

Usage Notes

Upon successful completion, `ldap_result()` returns the type of the first result returned in the res parameter. This is one of the following constants.

```
LDAP_RES_BIND (0x61)

LDAP_RES_SEARCH_ENTRY (0x64)

LDAP_RES_SEARCH_REFERENCE (0x73)
```
-- new in LDAPv3

```
LDAP_RES_SEARCH_RESULT (0x65)

LDAP_RES_MODIFY (0x67)

LDAP_RES_ADD (0x69)

LDAP_RES_DELETE (0x6B)

LDAP_RES_MODDN (0x6D)
```

```
LDAP_RES_COMPARE (0x6F)
```

`LDAP_RES_EXTENDED (0x78)` -- new in LDAPv3

`ldap_result()` returns `0` if the timeout expired and `-1` if an error occurs, in which case the error parameters of the LDAP session handle is set accordingly.

`ldap_msgfree()` frees each message in the result chain pointed to by `res` and returns the type of the last message in the chain. If `res` is `NULL`, then nothing is done and the value zero is returned.

`ldap_msgtype`) returns the type of the LDAP message it is passed as a parameter. The type is one of the types listed previously, or `-1` on error.

`ldap_msgid()` returns the message ID associated with the LDAP message passed as a parameter, or `-1` on error.

8.17 Handling Errors and Parsing Results

You can extract information from results and handle errors returned by other LDAP API routines. Note that `ldap_parse_sasl_bind_result()` and `ldap_parse_extended_result()` must typically be used in addition to `ldap_parse_result()` to retrieve all the result information from SASL Bind and Extended Operations respectively.

The `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, and `ldap_parse_extended_result()` functions all skip over messages of type `LDAP_RES_SEARCH_ENTRY` and `LDAP_RES_SEARCH_REFERENCE` when looking for a result message to parse. They return the constant `LDAP_SUCCESS` if the result was successfully parsed and another LDAP error code if not. Note that the LDAP error code that indicates the outcome of the operation performed by the server is placed in the errcodep `ldap_parse_result()` parameter. If a chain of messages that contains more than one result message is passed to these routines they always operate on the first result in the chain.

`ldap_err2string()` is used to convert a numeric LDAP error code, as returned by `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, `ldap_parse_extended_result()` or one of the synchronous API operation calls, into an informative zero-terminated character string message describing the error. It returns a pointer to static data.

8.17.1 ldap_parse_result, ldap_parse_extended_result, ldap_result2error

The functions `ldap_parse_result`,`ldap_parse_extended_result` and `ldap_result2error` are used to handle errors.

Syntax

```
int ldap_parse_result
(


);
LDAP            *ld,
LDAPMessage     *res,
int             *errcodep,
char            **matcheddnp,
char            **errmsgp,
char            ***referralsp,
```

```
LDAPControl      ***serverctrlsp,
int              freeit

int ldap_parse_sasl_bind_result
(

LDAP            *ld,
LDAPMessage     *res,
struct berval   **servercredp,
int              freeit

);


int ldap_parse_extended_result
(

LDAP            *ld,
LDAPMessage     *res,
char            **retoidp,
struct berval   **retdatap,
int              freeit

);
#define LDAP_NOTICE_OF_DISCONNECTION    "1.3.6.1.4.1.1466.20036"
char *ldap_err2string( int err );
```

The routines immediately following are deprecated. To learn more about them, see RFC 1823.

```
int ldap_result2error
(

LDAP            *ld,
LDAPMessage     *res,
int              freeit

);
void ldap_perror( LDAP *ld, const char *msg );
```

Parameters

Table 8-23 lists and describes parameters for handling errors and parsing results.

Table 8-23   Parameters for Handling Errors and Parsing Results Functions

Parameter	Description
ld	The session handle.
res	The result of an LDAP operation as returned by ldap_result() or one of the synchronous API operation calls.
errcodep	This result parameter is filled in with the LDAP error code field from the LDAPMessage message. This is the indication from the server of the outcome of the operation. NULL should be passed to ignore this field.

Table 8-23   (Cont.) Parameters for Handling Errors and Parsing Results Functions

Parameter	Description
matcheddnp	In the case of a return of `LDAP_NO_SUCH_OBJECT`, this result parameter is filled in with a DN indicating how much of the name in the request was recognized. `NULL` should be passed to ignore this field. The matched DN string should be freed by calling `ldap_memfree()` which is described later in this document.
errmsgp	This result parameter is filled in with the contents of the error message field from the `LDAPMessage` message. The error message string should be freed by calling `ldap_memfree()` which is described later in this document. `NULL` should be passed to ignore this field.
referralsp	This result parameter is filled in with the contents of the referrals field from the `LDAPMessage` message, indicating zero or more alternate LDAP servers where the request is to be retried. The referrals array should be freed by calling `ldap_value_free()` which is described later in this document. `NULL` should be passed to ignore this field.
serverctrlsp	This result parameter is filled in with an allocated array of controls copied out of the `LDAPMessage` message. The control array should be freed by calling `ldap_controls_free()` which was described earlier.
freeit	A Boolean that determines whether the `res` parameter is disposed of or not. Pass any nonzero value to have these routines free `res` after extracting the requested information. This is provided as a convenience; you can also use `ldap_msgfree()` to free the result later. If `freeit` is nonzero, the entire chain of messages represented by res is disposed of.
servercredp	For SASL bind results, this result parameter is filled in with the credentials passed back by the server for mutual authentication, if given. An allocated `berval` structure is returned that should be disposed of by calling `ber_bvfree()`. `NULL` should be passed to ignore this field.
retoidp	For extended results, this result parameter is filled in with the dotted-OID text representation of the name of the extended operation response. This string should be disposed of by calling `ldap_memfree()`. `NULL` should be passed to ignore this field. The `LDAP_NOTICE_OF_DISCONNECTION` macro is defined as a convenience for clients that wish to check an OID to see if it matches the one used for the unsolicited Notice of Disconnection (defined in RFC 2251[2] section 4.4.1).
retdatap	For extended results, this result parameter is filled in with a pointer to a `struct berval` containing the data in the extended operation response. It should be disposed of by calling `ber_bvfree()`. `NULL` should be passed to ignore this field.
err	For `ldap_err2string()`, an LDAP error code, as returned by `ldap_parse_result()` or another LDAP API call.

Usage Notes

See RFC 1823 for a description of parameters peculiar to the deprecated routines.

8.18 Stepping Through a List of Results

The `ldap_result()` is used to return the list of messages in a result chain.

The result chain for search operations can include referral messages, entry messages, and result messages.

`ldap_count_messages()` is used to count the number of messages returned. The `ldap_msgtype()` function, described previously, can be used to distinguish between the different message types.

8.18.1 ldap_first_message, ldap_next_message, ldap_count_messages

`ldap_first_message()` and `ldap_next_message()` returns `NULL` when no more messages exist in the result set to be returned.

Syntax

```
LDAPMessage *ldap_first_message( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_message( LDAP *ld, LDAPMessage *msg );
int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters

Table 8-24 lists and describes the parameters for stepping through a list of results.

Table 8-24   Parameters for Stepping Through a List of Results

Parameter	Description
ld	The session handle.
res	The result chain, as obtained by a call to one of the synchronous search routines or `ldap_result()`.
msg	The message returned by a previous call to `ldap_first_message()` or `ldap_next_message()`.

Usage Notes

`NULL` is returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle ld is set to indicate the error.

If successful, `ldap_count_messages()` returns the number of messages contained in a chain of results; if an error occurs such as the res parameter being invalid, `-1` is returned. The `ldap_count_messages()` call can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, `ldap_next_reference()`.

8.19 Parsing Search Results

You can parse the entries and references returned by `ldap_search` functions.

These results are returned in an opaque structure that may be accessed by calling the routines described in this section. Routines are provided to step through the entries and references returned, step through the attributes of an entry, retrieve the name of an entry, and retrieve the values associated with a given attribute in an entry. The functions are described in the following sections:

- ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_count_entries, and ldap_count_references
- ldap_first_attribute and ldap_next_attribute
- ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, and ldap_value_free_len
- ldap_get_dn, ldap_explode_dn, ldap_explode_rdn, and ldap_dn2ufn
- ldap_get_entry_controls
- ldap_parse_reference

8.19.1 ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_count_entries, and ldap_count_references

The `ldap_first_entry()` and `ldap_next_entry()` routines are used to step through and retrieve the list of entries from a search result chain.

The `ldap_first_reference()` and `ldap_next_reference()` routines are used to step through and retrieve the list of continuation references from a search result chain. `ldap_count_entries()` is used to count the number of entries returned. `ldap_count_references()` is used to count the number of references returned.

Syntax

```
LDAPMessage *ldap_first_entry( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_entry( LDAP *ld, LDAPMessage *entry );
LDAPMessage *ldap_first_reference( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_reference( LDAP *ld, LDAPMessage *ref );
int ldap_count_entries( LDAP *ld, LDAPMessage *res );
int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters

Table 8-25 lists and describes the parameters or retrieving entries and continuation references from a search result chain, and for counting entries returned.

Table 8-25   Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
ld	The session handle.

Table 8-25   (Cont.) Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
res	The search result, as obtained by a call to one of the synchronous search routines or `ldap_result()`.
entry	The entry returned by a previous call to `ldap_first_entry()` or `ldap_next_entry()`.
ref	The reference returned by a previous call to `ldap_first_reference()` or `ldap_next_reference()`.

Usage Notes

`ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, and `ldap_next_reference()` all return `NULL` when no more entries or references exist in the result set to be returned. `NULL` is also returned if an error occurs while stepping through the entries or references, in which case the error parameters in the session handle `ld` is set to indicate the error.

`ldap_count_entries()` returns the number of entries contained in a chain of entries; if an error occurs such as the `res` parameter being invalid, `-1` is returned. The `ldap_count_entries()` call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, `ldap_next_reference()`.

`ldap_count_references()` returns the number of references contained in a chain of search results; if an error occurs such as the `res` parameter being invalid, `-1` is returned. The `ldap_count_references()` call can also be used to count the number of references that remain in a chain.

8.19.2 ldap_first_attribute and ldap_next_attribute

`ldap_first_attribute()` and `ldap_next_attribute()` returns NULL when the end of the attributes is reached, or if there is an error. In the latter case, the error parameters in the session handle ld are set to indicate the error.

Use the functions in this section to step through the list of attribute types returned with an entry.

Syntax

```
char *ldap_first_attribute
(

LDAP            *ld,
LDAPMessage     *entry,
BerElement      **ptr

);

char *ldap_next_attribute
(
```

```
LDAP            *ld,
LDAPMessage     *entry,
BerElement      *ptr

);
void ldap_memfree( char *mem );
```

Parameters

Table 8-26 lists and describes the parameters for stepping through attribute types returned with an entry.

Table 8-26   Parameters for Stepping Through Attribute Types Returned with an Entry

Parameter	Description
ld	The session handle.
entry	The entry whose attributes are to be stepped through, as returned by ldap_first_entry() or ldap_next_entry().
ptr	In ldap_first_attribute(), the address of a pointer used internally to keep track of the current position in the entry. In ldap_next_attribute(), the pointer returned by a previous call to ldap_first_attribute(). The BerElement type itself is an opaque structure.
mem	A pointer to memory allocated by the LDAP library, such as the attribute type names returned by ldap_first_attribute() and ldap_next_attribute, or the DN returned by ldap_get_dn(). If mem is NULL, the ldap_memfree() call does nothing.

Usage Notes

ldap_first_attribute() and ldap_next_attribute() routines return a pointer to an allocated buffer containing the current attribute name. This should be freed when no longer in use by calling ldap_memfree().

ldap_first_attribute() allocates and returns in ptr a pointer to a BerElement used to keep track of the current position. This pointer may be passed in subsequent calls to ldap_next_attribute() to step through the entry's attributes. After a set of calls to ldap_first_attribute() and ldap_next_attribute(), if ptr is non-null, it should be freed by calling ber_free(ptr, 0). Note that it is very important to pass the second parameter as 0 (zero) in this call, since the buffer associated with the BerElement does not point to separately allocated memory.

The attribute type names returned are suitable for passing in a call to ldap_get_values() and friends to retrieve the associated values.

8.19.3 ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, and ldap_value_free_len

ldap_get_values() and ldap_get_values_len() are used to retrieve the values of a given attribute from an entry. ldap_count_values() and ldap_count_values_len() are used to count the returned values.

ldap_value_free() and ldap_value_free_len() are used to free the values.

Syntax

```
char **ldap_get_values
(

LDAP            *ld,
LDAPMessage     *entry,
const char      *attr

);

struct berval **ldap_get_values_len
(

LDAP            *ld,
LDAPMessage     *entry,
const char      *attr

);

int ldap_count_values( char **vals );
int ldap_count_values_len( struct berval **vals );
void ldap_value_free( char **vals );
void ldap_value_free_len( struct berval **vals );
```

Parameters

Table 8-27 lists and describes the parameters for retrieving and counting attribute values.

Table 8-27   Parameters for Retrieving and Counting Attribute Values

Parameter	Description
ld	The session handle.
entry	The entry from which to retrieve values, as returned by ldap_first_entry() or ldap_next_entry().
attr	The attribute whose values are to be retrieved, as returned by ldap_first_attribute() or ldap_next_attribute(), or a caller-supplied string (for example, "mail").
vals	The values returned by a previous call to ldap_get_values() or ldap_get_values_len().

Usage Notes

Two forms of the various calls are provided. The first form is only suitable for use with non-binary character string data. The second _len form is used with any kind of data.

ldap_get_values() and ldap_get_values_len() return NULL if no values are found for attr or if an error occurs.

ldap_count_values() and ldap_count_values_len() return -1 if an error occurs such as the vals parameter being invalid.

If a NULL vals parameter is passed to ldap_value_free() or ldap_value_free_len(), nothing is done.

Note that the values returned are dynamically allocated and should be freed by calling either `ldap_value_free()` or `ldap_value_free_len()` when no longer in use.

8.19.4 ldap_get_dn, ldap_explode_dn, ldap_explode_rdn, and ldap_dn2ufn

`ldap_get_dn()` is used to retrieve the name of an entry. `ldap_explode_dn()` and `ldap_explode_rdn()` are used to break up a name into its component parts. `ldap_dn2ufn()` is used to convert the name into a more user friendly format.

Syntax

```
char *ldap_get_dn( LDAP *ld, LDAPMessage *entry );
char **ldap_explode_dn( const char *dn, int notypes );
char **ldap_explode_rdn( const char *rdn, int notypes );
char *ldap_dn2ufn( const char *dn );
```

Parameters

Table 8-28 lists and describes the parameters for retrieving, exploding, and converting entry names.

Table 8-28   Parameters for Retrieving, Exploding, and Converting Entry Names

Parameter	Description
ld	The session handle.
entry	The entry whose name is to be retrieved, as returned by `ldap_first_entry()` or `ldap_next_entry()`.
dn	The DN to explode, such as returned by `ldap_get_dn()`.
rdn	The RDN to explode, such as returned in the components of the array returned by `ldap_explode_dn()`.
notypes	A Boolean parameter, if nonzero indicating that the DN or RDN components are to have their type information stripped off: `cn=Babs` would become `Babs`.

Usage Notes

`ldap_get_dn()` returns `NULL` if a DN parsing error occurs. The function sets error parameters in the session handle `ld` to indicate the error. It returns a pointer to newly allocated space that the caller should free by calling `ldap_memfree()` when it is no longer in use.

`ldap_explode_dn()` returns a `NULL`-terminated `char *` array containing the RDN components of the DN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the DN. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_explode_rdn()` returns a `NULL`-terminated `char *` array containing the components of the RDN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the rdn. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_dn2ufn()` converts the DN into a user friendly format. The UFN returned is newly allocated space that should be freed by a call to `ldap_memfree()` when no longer in use.

8.19.5 ldap_get_entry_controls

`ldap_get_entry_controls()` is used to extract LDAP controls from an entry.

Syntax

```
int ldap_get_entry_controls
(

LDAP            *ld,
LDAPMessage     *entry,
LDAPControl     ***serverctrlsp

);
```

Parameters

Table 8-29 lists and describes the parameters for extracting LDAP control from an entry.

Table 8-29   Parameters for Extracting LDAP Controls from an Entry

Parameters	Description
ld	The session handle.
entry	The entry to extract controls from, as returned by `ldap_first_entry()` or `ldap_next_entry()`.
serverctrlsp	This result parameter is filled in with an allocated array of controls copied out of entry. The control array should be freed by calling `ldap_controls_free()`. If `serverctrlsp` is `NULL`, no controls are returned.

Usage Notes

`ldap_get_entry_controls()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

8.19.6 ldap_parse_reference

Use `ldap_parse_reference()` to extract referrals and controls from a `SearchResultReference` message

Syntax

```
int ldap_parse_reference
(

LDAP            *ld,
LDAPMessage     *ref,
char            ***referralsp,
LDAPControl     ***serverctrlsp,
int             freeit

);
```

Parameters

Table 8-30 lists and describes parameters for extracting referrals and controls from a `SearchResultReference` message.

Table 8-30   Parameters for Extracting Referrals and Controls from a SearchResultReference Message

Parameter	Description
ld	The session handle.
ref	The reference to parse, as returned by `ldap_result()`, `ldap_first_reference()`, or `ldap_next_reference()`.
referralsp	This result parameter is filled in with an allocated array of character strings. The elements of the array are the referrals (typically LDAP URLs) contained in ref. The array should be freed when no longer in used by calling `ldap_value_free()`. If `referralsp` is `NULL`, the referral URLs are not returned.
serverctrlsp	This result parameter is filled in with an allocated array of controls copied out of `ref`. The control array shouldbe freed by calling `ldap_controls_free()`. If `serverctrlsp` is `NULL`, no controls are returned.
freeit	A Boolean that determines whether the `ref` parameter is disposed of or not. Pass any nonzero value to have this routine free `ref` after extracting the requested information. This is provided as a convenience. You can also use `ldap_msgfree()` to free the result later.

Usage Notes

`ldap_parse_reference()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

9

DBMS_LDAP PL/SQL Reference

`DBMS_LDAP` contains the functions and procedures that enable PL/SQL programmers to access data from LDAP servers.
The following topics describe DBMS_LDAP package:

- API Subprograms of a DBMS_LDAP Package
- Exception Summary of a DBMS_LDAP Package
- Data Types Used by DBMS_LDAP Package

9.1 API Subprograms of a DBMS_LDAP Package

API Subprograms of a DBMS_LDAP Package include the functions to perform establishing a session, authenticating a user, searching in the directory etc.

The following table describes the DBMS_LDAP API subprograms:

Table 9-1   DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION init	`init()` initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.
FUNCTION simple_bind_s	The function `simple_bind_s()` can be used to perform simple user name and password authentication to the directory server.
FUNCTION bind_s	The function `bind_s()` can be used to perform complex authentication to the directory server.
FUNCTION unbind_s	The function `unbind_s()` is used for closing an active LDAP session.
FUNCTION compare_s	The function `compare_s()` can be used to test if a particular attribute in a particular entry has a particular value.
FUNCTION search_s	The function `search_s()` performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out by the server.
FUNCTION search_st	The function `search_st()` performs a synchronous search in the LDAP server with a client side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the client or the server.
FUNCTION first_entry	The function first_entry is used to retrieve the first entry in the result set returned by either `search_s()` or search_st.
FUNCTION next_entry	The function `next_entry()` is used to iterate to the next entry in the result set of a search operation.

Table 9-1   (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION count_entries	This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions `first_entry()` and `next_entry`.
FUNCTION first_attribute	The function `first_attribute()` fetches the first attribute of a given entry in the result set.
FUNCTION next_attribute	The function `next_attribute()`fetches the next attribute of a given entry in the result set.
FUNCTION get_dn	The function `get_dn()` retrieves the X.500 distinguished name of a given entry in the result set.
FUNCTION get_values	The function `get_values()`can be used to retrieve all of the values associated with a given attribute in a given entry.
FUNCTION get_values_len	The function `get_values_len()` can be used to retrieve values of attributes that have a 'Binary' syntax.
FUNCTION delete_s	This function can be used to remove a leaf entry in the LDAP Directory Information Tree.
FUNCTION modrdn2_s	The function `modrdn2_s()` can be used to rename the relative distinguished name of an entry.
FUNCTION err2string	The function `err2string()` can be used to convert an LDAP error code to a string in the local language in which the API is operating.
FUNCTION create_mod_array	The function `create_mod_array()` allocates memory for array modification entries that are applied to an entry using the `modify_s()` functions.
PROCEDURE populate_mod_array (String Version)	Populates one set of attribute information for add or modify operations. This procedure call has to happen after `DBMS_LDAP.create_mod_array()` is called.
PROCEDURE populate_mod_array (Binary Version)	Populates one set of attribute information for add or modify operations. This procedure call has to occur after `DBMS_LDAP.create_mod_array()` is called.
PROCEDURE populate_mod_array (Binary Version. Uses BLOB Data Type)	Populates one set of attribute information for add or modify operations. This procedure call has to happen after `DBMS_LDAP.create_mod_array()` is called.
FUNCTION get_values_blob	The function `get_values_blob()` can be used to retrieve larger values of attributes that have a binary syntax.
FUNCTION count_values_blob	Counts the number of values returned by `DBMS_LDAP.get_values_blob()`.
FUNCTION value_free_blob	Frees the memory associated with the `BLOB_COLLECTION` returned by `DBMS_LDAP.get_values_blob()`.
FUNCTION modify_s	Performs a synchronous modification of an existing LDAP directory entry. Before calling `add_s`, you must call `DBMS_LDAP.creat_mod_array()` and `DBMS_LDAP.populate_mod_array()`.
FUNCTION add_s	Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, you must call `DBMS_LDAP.creat_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

Table 9-1   (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
PROCEDURE free_mod_array	Frees the memory allocated by `DBMS_LDAP.create_mod_array()`.
FUNCTION count_values	Counts the number of values returned by `DBMS_LDAP.get_values()`.
FUNCTION count_values_len	Counts the number of values returned by `DBMS_LDAP.get_values_len ()`.
FUNCTION rename_s	Renames an LDAP entry synchronously.
FUNCTION explode_dn	Breaks a DN up into its components.
FUNCTION open_ssl	Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.
FUNCTION msgfree	This function frees the chain of messages associated with the message handle returned by synchronous search functions.
FUNCTION ber_free	This function frees the memory associated with a handle to `BER_ELEMENT`.
FUNCTION nls_convert_to_utf8	The `nls_convert_to_utf8` function converts the input string containing database character set data to UTF-8 character set data and returns it.
FUNCTION nls_convert_from_utf8	The `nls_convert_from_utf8` function converts the input string containing UTF-8 character set data to database character set data and returns it.
FUNCTION nls_get_dbcharset_name	The `nls_get_dbcharset_name` function returns a string containing the database character set name.

> ✏ **See Also:**
>
> - Searching the Directory for more about `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()`.
> - DBMS_LDAP API to Terminate a Session for more about `DBMS_LDAP.unbind_s()`.

9.2 Exception Summary of a DBMS_LDAP Package

You can have the exception summary of a DBMS_LDAP package which includes error messages raised for session, authentication and search mechanisms.

`DBMS_LDAP` can generate the exceptions described in Table 9-2.

Table 9-2   DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
`general_error`	31202	Raised anytime an error is encountered that does not have a specific PL/SQL exception associated with it. The error string contains the description of the problem in the user's language.
`init_failed`	31203	Raised by `DBMS_LDAP.init()` if there are problems.
`invalid_session`	31204	Raised by all functions and procedures in the `DBMS_LDAP` package if they are passed an invalid session handle.
`invalid_auth_method d`	31205	Raised by `DBMS_LDAP.bind_s()` if the authentication method requested is not supported.
`invalid_search_scope pe`	31206	Raised by all search functions if the scope of the search is invalid.
`invalid_search_time e_val`	31207	Raised by `DBMS_LDAP.search_st()` if it is given an invalid value for a time limit.
`invalid_message`	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.
`count_entry_error`	31209	Raised by `DBMS_LDAP.count_entries` if it cannot count the entries in a given result set.
`get_dn_error`	31210	Raised by `DBMS_LDAP.get_dn` if the DN of the entry it is retrieving is `NULL`.
`invalid_entry_dn`	31211	Raised by all functions that modify, add, or rename an entry if they are presented with an invalid entry DN.
`invalid_mod_array`	31212	Raised by all functions that take a modification array as an argument if they are given an invalid modification array.
`invalid_mod_option`	31213	Raised by `DBMS_LDAP.populate_mod_array` if the modification option given is anything other than `MOD_ADD`, `MOD_DELETE` or `MOD_REPLACE`.
`invalid_mod_type`	31214	Raised by `DBMS_LDAP.populate_mod_array` if the attribute type that is being modified is `NULL`.
`invalid_mod_value`	31215	Raised by `DBMS_LDAP.populate_mod_array` if the modification value parameter for a given attribute is `NULL`.
`invalid_rdn`	31216	Raised by all functions and procedures that expect a valid RDN and are provided with an invalid one.

Table 9-2   (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
`invalid_newparent`	31217	Raised by `DBMS_LDAP.rename_s` if the new parent of an entry being renamed is `NULL`.
`invalid_deleteoldr dn`	31218	Raised by `DBMS_LDAP.rename_s` if the `deleteoldrdn` parameter is invalid.
`invalid_notypes`	31219	Raised by `DBMS_LDAP.explode_dn` if the `notypes` parameter is invalid.
`invalid_ssl_wallet _loc`	31220	Raised by `DBMS_LDAP.open_ssl` if the wallet location is `NULL` but the SSL authentication mode requires a valid wallet.
`invalid_ssl_wallet _password`	31221	Raised by `DBMS_LDAP.open_ssl` if the wallet password given is `NULL`.
`invalid_ssl_auth_m ode`	31222	Raised by `DBMS_LDAP.open_ssl` if the SSL authentication mode is not `1`, `2` or `3`.

9.3 Data Types Used by DBMS_LDAP Package

The `DBMS_LDAP` package uses the data types such as session, message, timeval etc.

Table 9-3   DBMS_LDAP Data Type Summary

Data-Type	Purpose
`SESSION`	Used to hold the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.
`MESSAGE`	Used to hold a handle to the message retrieved from the result set. This is used by all functions that work with entry attributes and values.
`MOD_ARRAY`	Used to hold a handle to the array of modifications being passed to either `modify_s()` or `add_s()`.
`TIMEVAL`	Used to pass time limit information to the LDAP API functions that require a time limit.
`BER_ELEMENT`	Used to hold a handle to a `BER` structure used for decoding incoming messages.
`STRING_COLLECTION`	Used to hold a list of `VARCHAR2` strings that can be passed on to the LDAP server.
`BINVAL_COLLECTION`	Used to hold a list of `RAW` data, which represent binary data.
`BERVAL_COLLECTION`	Used to hold a list of `BERVAL` values that are used for populating a modification array.
`BLOB_COLLECTION`	Used to hold a list of `BLOB` data, which represent binary data.

9.4 DBMS_LDAP API Subprograms

Each DBMS_LDAP API subprogram has different options and examples which uses functions and procedures.

Following are the list of DBMS_LDAP API Subprograms:

9.4.1 FUNCTION init

init() initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.

Syntax

```
FUNCTION init
(
hostname IN VARCHAR2,
portnum  IN PLS_INTEGER
)
RETURN SESSION;
```

Parameters

The following table describes init function parameters:

Table 9-4   INIT Function Parameters

Parameter	Description
hostname	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number, which is separated from the host by a colon. The hosts are tried in the order listed, stopping with the first one to which a successful connection is made.
portnum	Contains the TCP port number to connect to. If the port number is included with the host name, this parameter is ignored. If the parameter is not specified, and the host name does not contain the port number, a default port number of 3060 is assumed.

Return Values

The following table describes init function return values:

Table 9-5   INIT Function Return Values

Value	Description
SESSION	A handle to an LDAP session that can be used for further calls to the API.

Exceptions

The following table describes init function exceptions:

Table 9-6   INIT Function Exceptions

Exception	Description
init_failed	Raised when there is a problem contacting the LDAP server.
general_error	For all other errors. The error string associated with the exception describes the error in detail.

Usage Notes

DBMS_LDAP.init() is the first function that should be called because it establishes a session with the LDAP server. Function DBMS_LDAP.init() returns a session handle, a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. This routine returns NULL and raises the INIT_FAILED exception if the session cannot be initialized. After init() has been called, the connection has to be authenticated using DBMS_LDAP.bind_s or DBMS_LDAP.simple_bind_s().

> **Note:**
>
> - Ensure that the autonomous database nodes are able to resolve the LDAP server. See Private DNS to learn about how to manage private Domain Name System (DNS) zones, in case the LDAP server is in a different Virtual Cloud Network (VCN) than the autonomous database.
> - Open appropriate network firewall and security rules between LDAP and autonomous database nodes. See Security Rules.

Related Functions

DBMS_LDAP.simple_bind_s(), DBMS_LDAP.bind_s()

9.4.2 FUNCTION simple_bind_s

The function simple_bind_s can be used to perform simple user name and password authentication to the directory server.

Syntax

```
FUNCTION simple_bind_s
(
ld      IN SESSION,
dn      IN VARCHAR2,
passwd IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table describes the simple_bind_s function parameters:

Table 9-7   SIMPLE_BIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The Distinguished Name of the User that we are trying to login as.
passwd	A text string containing the password.

Return Values

The following table describe the simple_bind_s function return values:

Table 9-8   SIMPLE_BIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on a successful completion. If there was a problem, one of the following exceptions are raised.

Exceptions

The following table describes simple_bind_s function exceptions:

Table 9-9   SIMPLE_BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

DBMS_LDAP.simple_bind_s() can be used to authenticate a user whose directory distinguished name and directory password are known. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

9.4.3 FUNCTION bind_s

The function bind_s can be used to perform complex authentication to the directory server.

Syntax

```
FUNCTION  bind_s
(

ld     IN SESSION,
dn     IN VARCHAR2,
```

```
cred IN VARCHAR2,
meth IN PLS_INTEGER

)

RETURN PLS_INTEGER;
```

Parameters

The following table describes bind_s function parameters:

Table 9-10   BIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The distinguished name of the user.
cred	A text string containing the credentials used for authentication.
meth	The authentication method. The only valid value is DBMS_LDAP_UTL.AUTH_SIMPLE.

Return Values

The following table describes bind_s function return values:

Table 9-11   BIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS upon successful completion. One of the following exceptions is raised if there is a problem.

Exceptions

The following table describes bind_s function exception values:

Table 9-12   BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_auth_method	Raised if the authentication method requested is not supported.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

DBMS_LDAP.bind_s() can be used to authenticate a user. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

Related Functions

DBMS_LDAP.init(),DBMS_LDAP.simple_bind_s().

9.4.4 FUNCTION unbind_s

The function unbind_s is used for closing an active LDAP session.

Syntax

```
FUNCTION unbind_s
(

ld IN OUT SESSION

)

RETURN PLS_INTEGER;
```

Parameters

The following table describes UNBIND_S function parameters:

Table 9-13   UNBIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Return Values

The following table describes UNBIND_S function return values:

Table 9-14   UNBIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on proper completion. One of the following exceptions is raised otherwise.

Exceptions

The following table describes UNBIND_S function exceptions:

Table 9-15   UNBIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the sessions handle ld is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The `unbind_s()` function sends an unbind request to the server, closes all open connections associated with the LDAP session, and disposes of all resources associated with the session handle before returning. After a call to this function, the session handle `ld` is invalid.

Related Functions

`DBMS_LDAP.bind_s()`, `DBMS_LDAP.simple_bind_s()`.

9.4.5 FUNCTION compare_s

The function `compare_s` can be used to test if a particular attribute in a particular entry has a particular value.

Syntax

```
FUNCTION compare_s
(

ld    IN SESSION,
dn    IN VARCHAR2,
attr  IN VARCHAR2,
value IN VARCHAR2

)

RETURN PLS_INTEGER;
```

Parameters

The following table describes compare_s function parameters:

Table 9-16   COMPARE_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
value	A string attribute value to compare against.

Return Values

The following table describes compare_s function return values:

Table 9-17   COMPARE_S Function Return Values

Value	Description
PLS_INTEGER	COMPARE_TRUE if the given attribute has a matching value.
	COMPARE_FALSE if the given attribute does not have a matching value.

Exceptions

The following table lists compare_s function exceptions:

Table 9-18   COMPARE_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function compare_s can be used to assert that an attribute in the directory has a certain value. This operation can be performed only on attributes whose syntax enables them to be compared. The compare_s function can be called only after a valid LDAP session handle has been obtained from the init() function and authenticated by the bind_s() or simple_bind_s() functions.

Related Functions

DBMS_LDAP.bind_s().

9.4.6 FUNCTION search_s

The function search_s performs a synchronous search in the directory. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the server.

Syntax

```
FUNCTION search_s
(

ld       IN  SESSION,
base     IN  VARCHAR2,
scope    IN  PLS_INTEGER,
filter   IN  VARCHAR2,
attrs    IN  STRING_COLLECTION,
attronly IN  PLS_INTEGER,
res      OUT MESSAGE

)

RETURN PLS_INTEGER;
```

Parameters

The following table lists search_s function parameters:

Table 9-19   SEARCH_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The DN of the entry at which to start the search.
scope	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)", which matches all entries, is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS ("1.1") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
res	This is a result parameter that contains the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

The following table lists search_s function return values:

Table 9-20   SEARCH_S Function Return Value

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

The following table lists search_s function exceptions:

Table 9-21   SEARCH_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_search_scope	Raised if the search scope is not one of `SCOPE_BASE`, `SCOPE_ONELEVEL`, or `SCOPE_SUBTREE`.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function `search_s()` issues a search operation and does not return control to the user environment until all of the results have been returned from the server. Entries returned from the search, if any, are contained in the `res` parameter. This parameter is opaque to the caller. Entries, attributes, and values can be extracted by calling the parsing routines described in this chapter.

Related Functions

`DBMS_LDAP.search_st()`, `DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry`.

9.4.7 FUNCTION search_st

The function search_st() performs a synchronous search in the LDAP server with a client-side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the client or the server.

Syntax

```
FUNCTION search_st
(

ld       IN   SESSION,
base     IN   VARCHAR2,
scope    IN   PLS_INTEGER,
filter   IN   VARCHAR2,
attrs    IN   STRING_COLLECTION,
attronly IN   PLS_INTEGER,
tv       IN   TIMEVAL,
res      OUT  MESSAGE

)

RETURN PLS_INTEGER;
```

Parameters

The following function lists search_st function parameters:

Table 9-22   SEARCH_ST Function Parameters

Parameter	Description
`ld`	A valid LDAP session handle.
`base`	The DN of the entry at which to start the search.
`scope`	One of `SCOPE_BASE (0x00)`, `SCOPE_ONELEVEL (0x01)`, or `SCOPE_SUBTREE (0x02)`, indicating the scope of the search.
`filter`	A character string representing the search filter. The value `NULL` can be passed to indicate that the filter `"(objectclass=*)"`, which matches all entries, is to be used.
`attrs`	A collection of strings indicating which attributes to return for each matching entry. Passing `NULL` for this parameter causes all available user attributes to be retrieved. The special constant string `NO_ATTRS ("1.1")` may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string `ALL_USER_ATTRS ("*")` can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
`attrsonly`	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
`tv`	The time out value, expressed in seconds and microseconds, that should be used for this search.
`res`	This is a result parameter which contains the results of the search upon completion of the call. If no results are returned, `*res` is set to `NULL`.

Return Values

Following table lists search_st function return values:

Table 9-23   SEARCH_ST Function Return Values

Value	Description
`PLS_INTEGER`	`DBMS_LDAP.SUCCESS` if the search operation succeeded. An exception is raised in all other cases.
`res`	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

Following table lists search_st function exceptions:

Table 9-24   SEARCH_ST Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_val ue	Raised if the time value specified for the time out is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

This function is very similar to DBMS_LDAP.search_s() except that it requires a time out value to be given.

Related Functions

DBMS_LDAP.search_s(), DBML_LDAP.first_entry(), DBMS_LDAP.next_entry.

9.4.8 FUNCTION first_entry

The function first_entry() is used to retrieve the first entry in the result set returned by either search_s() or search_st().

Syntax

```
FUNCTION first_entry
(

ld  IN SESSION,
msg IN MESSAGE

)

RETURN MESSAGE;
```

Parameters

Following table lists first_entry function parameters:

Table 9-25   FIRST_ENTRY Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

The following table lists the first_entry function return values:

Table 9-26   FIRST_ENTRY Return Values

Value	Description
MESSAGE	A handle to the first entry in the list of entries returned from the LDAP server. It is set to NULL if there was an error and an exception is raised.

Exceptions

The following table lists first_entry exceptions:

Table 9-27   FIRST_ENTRY Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The function first_entry() should always be the first function used to retrieve the results from a search operation.

Related Functions

DBMS_LDAP.next_entry(), DBMS_LDAP.search_s(), DBMS_LDAP.search_st().

9.4.9 FUNCTION next_entry

This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions first_entry() and next_entry().

Syntax

```
FUNCTION count_entries
(

ld  IN SESSION,
msg IN MESSAGE

)

RETURN PLS_INTEGER;
```

Parameters

The following table count_entry function parameters:

Table 9-28   COUNT_ENTRY Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

The following table lists count_entry function return values:

Table 9-29   COUNT_ENTRY Function Return Values

Value	Description
PLS_INTEGER	Nonzero if there are entries in the result set. -1 if there was a problem.

Exceptions

The following table lists count_entry function exceptions:

Table 9-30   COUNT_ENTRY Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.
count_entry_error	Raised if there was a problem in counting the entries.

Usage Notes

count_entries() returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid, -1 is returned. The count_entries() call can also be used to count the number of entries that remain in a chain if called with a message, entry, or reference returned by first_message(), next_message(), first_entry(), next_entry(), first_reference(), next_reference().

Related Functions

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

9.4.10 FUNCTION count_entries

The function `first_attribute()` fetches the first attribute of a given entry in the result set.

Syntax

```
FUNCTION first_attribute
(

ld          IN  SESSION,
ldapentry   IN  MESSAGE,
ber_elem    OUT BER_ELEMENT

)

RETURN VARCHAR2;
```

Parameters

The following table lists first_attribute function parameters:

Table 9-31   FIRST_ATTRIBUTE Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose attributes are to be stepped through, as returned by `first_entry()` or `next_entry()`.
ber_elem	A handle to a `BER_ELEMENT` that is used to keep track of attributes in the entry that have already been read.

Return Values

The following table lists first attribute function return values:

Table 9-32   FIRST_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2	The name of the attribute if it exists. `NULL` if no attribute exists or if an error occurred.
ber_elem	A handle used by `DBMS_LDAP.next_attribute()` to iterate over all of the attributes

Exceptions

The following table lists first_attribue function exceptions:

Table 9-33   FIRST_ATTRIBUTE Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_message	Raised if the incoming `msg` handle is invalid.

Usage Notes

The handle to the `BER_ELEMENT` returned as a function parameter to `first_attribute()` should be used in the next call to `next_attribute()` to iterate through the various attributes of an entry. The name of the attribute returned from a call to `first_attribute()` can in turn be used in calls to the functions `get_values()` or `get_values_len()` to get the values of that particular attribute.

Related Functions

`DBMS_LDAP.next_attribute()`, `DBMS_LDAP.get_values()`, `DBMS_LDAP.get_values_len()`, `DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`.

9.4.11 FUNCTION first_attribute

The function `next_attribute()` retrieves the next attribute of a given entry in the result set.

Syntax

```
FUNCTION next_attribute
(

ld        IN SESSION,
ldapentry      IN MESSAGE,
ber_elem IN BER_ELEMENT

)

RETURN VARCHAR2;
```

Parameters

The following table describes the next_attribute function parameters:

Table 9-34   NEXT_ATTRIBUTE Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose attributes are to be stepped through, as returned by `first_entry()` or `next_entry()`.
ber_elem	A handle to a `BER_ELEMENT` that is used to keep track of attributes in the entry that have been read.

Return Values

The following table describes next_attribute function return values:

Table 9-35   NEXT_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2 (function return)	The name of the attribute if it exists.

Exceptions

The following table describes next_attribute function exceptions:

Table 9-36   NEXT_ATTRIBUTE Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute() should be used in the next call to next_attribute() to iterate through the various attributes of an entry. The name of the attribute returned from a call to next_attribute() can in turn be used in calls to the functions get_values() or get_values_len() to get the values of that particular attribute.

Related Functions

DBMS_LDAP.first_attribute(), DBMS_LDAP.get_values(), DBMS_LDAP.get_values_len(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

9.4.12 FUNCTION next_attribute

The function get_dn() retrieves the X.500 distinguished name of given entry in the result set.

Syntax

```
FUNCTION get_dn
(

ld  IN SESSION,
ldapentrymsg IN MESSAGE

)

RETURN VARCHAR2;
```

Parameters

The following table describes get_dn function parameters:

Table 9-37   GET_DN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose DN is to be returned.

Return Values

The following function describes get_dn function return values:

Table 9-38   GET_DN Function Return Values

Value	Description
VARCHAR2	The X.500 Distinguished name of the entry as a PL/SQL string. NULL if there was a problem.

Exceptions

The following function describes get_dn function exceptions:

Table 9-39   GET_DN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN.

Usage Notes

The function get_dn() can be used to retrieve the DN of an entry as the program logic is iterating through the result set. This can in turn be used as an input to explode_dn() to retrieve the individual components of the DN.

Related Functions

DBMS_LDAP.explode_dn().

9.4.13 FUNCTION get_dn

The function `get_values()` can be used to retrieve all of the values associated with a given attribute in a given entry.

Syntax

```
FUNCTION get_values
(

ld   IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2

)

RETURN STRING_COLLECTION;
```

Parameters

Following table describes get_values function parameters:

Table 9-40   GET_VALUES Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	A valid handle to an entry returned from a search result.
attr	The name of the attribute for which values are being sought.

Return Values

Following table describes get_values function return values:

Table 9-41   GET_VALUES Function Return Values

Value	Description
STRING_COLLECTION	A PL/SQL string collection containing all of the values of the given attribute.
	NULL if there are no values associated with the given attribute.

Exceptions

Following function describes get_values function exceptions:

Table 9-42   GET_VALUES Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function `get_values()` can only be called after the handle to entry has been first retrieved by call to either `first_entry()` or `next_entry()`. The name of the attribute may be known beforehand or can be determined by a call to `first_attribute()` or `next_attribute()`.The function `get_values()` always assumes that the data type of the attribute it is retrieving is a string. For retrieving binary data types, `get_values_len()` should be used.

Related Functions

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`, `DBMS_LDAP.count_values()`, `DBMS_LDAP.get_values_len()`.

9.4.14 FUNCTION get_values

The function `get_values()` can be used to retrieve all of the values associated with a given attribute in a given entry.

Syntax

```
FUNCTION get_values
(

ld   IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2

)

RETURN STRING_COLLECTION;
```

Parameters

Following table describes get_values function parameters:

Table 9-43   GET_VALUES Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	A valid handle to an entry returned from a search result.

Table 9-43   (Cont.) GET_VALUES Function Parameters

Parameter	Description
attr	The name of the attribute for which values are being sought.

Return Values

Following table describes get_values function return values:

Table 9-44   GET_VALUES Function Return Values

Value	Description
STRING_COLLECTION	A PL/SQL string collection containing all of the values of the given attribute.
	NULL if there are no values associated with the given attribute.

Exceptions

Following function describes get_values function exceptions:

Table 9-45   GET_VALUES Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values() can only be called after the handle to entry has been first retrieved by call to either first_entry() or next_entry(). The name of the attribute may be known beforehand or can be determined by a call to first_attribute() or next_attribute().The function get_values() always assumes that the data type of the attribute it is retrieving is a string. For retrieving binary data types, get_values_len() should be used.

Related Functions

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

9.4.15 FUNCTION get_values_len

The function `get_values_len()` can be used to retrieve values of attributes that have a binary syntax.

Syntax

```
FUNCTION get_values_len
(

ld    IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2

)

RETURN BINVAL_COLLECTION;
```

Parameters

Following function describes get_values_len function parameters:

Table 9-46   GET_VALUES_LEN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.
attr	The string name of the attribute for which values are being sought.

Return Values

Following table describes get_values_len function return values:

Table 9-47   GET_VALUES_LEN Function Return Values

Value	Description
BINVAL_COLLECTION	A PL/SQL 'Raw' collection containing all the values of the given attribute.
	NULL if there are no values associated with the given attribute.

Exceptions

Following table describes get_values_len function exceptions:

Table 9-48   GET_VALUES_LEN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values_len() can only be called after the handle to an entry has been retrieved by a call to either first_entry() or next_entry().The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute().This function can be used to retrieve both binary and non-binary attribute values.

Related Functions

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values_len(), DBMS_LDAP.get_values().

9.4.16 **FUNCTION get_values_len**

The function get_values_len() can be used to retrieve values of attributes that have a binary syntax.

Syntax

```
FUNCTION get_values_len
(

ld   IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2

)

RETURN BINVAL_COLLECTION;
```

Parameters

Following function describes get_values_len function parameters:

Table 9-49   GET_VALUES_LEN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.
attr	The string name of the attribute for which values are being sought.

Return Values

Following table describes get_values_len function return values:

Table 9-50   GET_VALUES_LEN Function Return Values

Value	Description
BINVAL_COLLECTION	A PL/SQL 'Raw' collection containing all the values of the given attribute.
	NULL if there are no values associated with the given attribute.

Exceptions

Following table describes get_values_len function exceptions:

Table 9-51   GET_VALUES_LEN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values_len() can only be called after the handle to an entry has been retrieved by a call to either first_entry() or next_entry().The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute().This function can be used to retrieve both binary and non-binary attribute values.

Related Functions

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(),
DBMS_LDAP.count_values_len(), DBMS_LDAP.get_values().

9.4.17 FUNCTION delete_s

The function delete_s() can be used to remove a leaf entry in the DIT.

Parameters

Return Values

Exceptions

Usage Notes

See Also

Syntax

```
FUNCTION delete_s
(
```

```
ld      IN SESSION,
entrydn IN VARCHAR2

)

RETURN PLS_INTEGER;
```

Parameters

Following table describes delete_s function parameters:

Table 9-52   DELETE_S Function Parameters

Parameter Name	Description
ld	A valid LDAP session.
entrydn	The X.500 distinguished name of the entry to delete.

Return Values

Following table describes delete_s function return values:

Table 9-53   DELETE_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the delete operation was successful. An exception is raised otherwise.

Exceptions

Following table describes delete_s function exceptions:

Table 9-54   DELETE_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function delete_s() can be used to remove only leaf entries in the DIT. A leaf entry is an entry that does not have any entries under it. This function cannot be used to delete non-leaf entries.

Related Functions

DBMS_LDAP.modrdn2_s().

9.4.18 FUNCTION modrdn2_s

The function `modrdn2_s()` can be used to rename the relative distinguished name of an entry.

Syntax

```
FUNCTION modrdn2_s
(

ld IN SESSION,
entrydn in VARCHAR2
newrdn in VARCHAR2
deleteoldrdn IN PLS_INTEGER

)

RETURN PLS_INTEGER;
```

Parameters

The following table describes modrdn2_s function parameters:

Table 9-55   MODRDN2_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
entrydn	The distinguished name of the entry (This entry must be a leaf node in the DIT.).
newrdn	The new relative distinguished name of the entry.
deleteoldrdn	A boolean value that, if nonzero, indicates that the attribute values from the old name should be removed from the entry.

Return Values

The following table describes modrdn2_s function return values:

Table 9-56   MODRDN2_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.

Exceptions

The following table describes modrdn2_s function exceptions:

Table 9-57   MODRDN2_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
invalid_rdn	Invalid LDAP RDN.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.
general_error	For all other errors. The error string associated with this exception explains the error in detail.

Usage Notes

The function nodrdn2_s() can be used to rename the leaf nodes of a DIT. It simply changes the relative distinguished name by which they are known. The use of this function is being deprecated in the LDAP v3 standard. Please use rename_s(), which fulfills the same purpose.

Related Functions

DBMS_LDAP.rename_s().

9.4.19 FUNCTION err2string

The function err2string() can be used to convert an LDAP error code to a string in the local language in which the API is operating.

Syntax

```
FUNCTION err2string
(

ldap_err IN PLS_INTEGER

)

RETURN VARCHAR2;
```

Parameters

The following table describes err2string function parameters:

Table 9-58   ERR2STRING Function Parameters

Parameter	Description
ldap_err	An error number returned from one of the API calls.

Return Values

The following table describes err2string function return values:

Table 9-59   ERR2STRING Function Return Values

Value	Description
VARCHAR2	A character string translated to the local language. The string describes the error in detail.

Exceptions

err2string() raises no exceptions.

Usage Notes

In this release, the exception handling mechanism automatically invokes this function if any of the API calls encounter an error.

Related Functions

None

9.4.20 FUNCTION create_mod_array

The function create_mod_array() allocates memory for array modification entries that are applied to an entry using the modify_s() or add_s() functions.

Syntax

```
FUNCTION create_mod_array
(
num IN PLS_INTEGER
)
RETURN MOD_ARRAY;
```

Parameters

The following table describes create_mod_array function parameters:

Table 9-60   CREATE_MOD_ARRAY Function Parameters

Parameter	Description
num	The number of the attributes that you want to add or modify.

Return Values

The following table describes create_mod_array function return values:

Table 9-61   CREATE_MOD_ARRAY Function Return Values

Value	Description
MOD_ARRAY	The data structure holds a pointer to an LDAP mod array.
	Returns NULL if there was a problem.

Exceptions

create_mod_array() raises no exceptions.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It calls DBMS_LDAP.free_mod_array to free memory after the calls to add_s or modify_s have completed.

Related Functions

DBMS_LDAP.populate_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

9.4.21 PROCEDURE populate_mod_array (String Version)

Populates one set of attribute information for add or modify operations.

Syntax

```
PROCEDURE populate_mod_array
(

modptr   IN DBMS_LDAP.MOD_ARRAY,
mod_op   IN PLS_INTEGER,
mod_type IN VARCHAR2,
modval   IN DBMS_LDAP.STRING_COLLECTION

);
```

Parameters

Following table describes populate_mod_array procedure parameters:

Table 9-62   POPULATE_MOD_ARRAY (String Version) Procedure Parameters

Parameter	Description
modptr	The data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform, MOD_ADD, MOD_DELETE or MOD_REPLACE.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modval	This field specifies the attribute values to add, delete, or replace. It is for string values only.

Exceptions

Following table describes populate_mod_array procedure exceptions:

Table 9-63   POPULATE_MOD_ARRAY (String Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array
invalid_mod_option	Invalid LDAP mod option
invalid_mod_type	Invalid LDAP mod type
invalid_mod_value	Invalid LDAP mod value

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It has to happen after `DBMS_LDAP.create_mod_array` is called.

Related Functions

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

9.4.22 PROCEDURE populate_mod_array (Binary Version)

Populates one set of attribute information for add or modify operations. This procedure call occurs after `DBMS_LDAP.create_mod_array()` is called.

Syntax

```
PROCEDURE populate_mod_array
(

modptr   IN DBMS_LDAP.MOD_ARRAY,
mod_op   IN PLS_INTEGER,
mod_type IN VARCHAR2,
modbval  IN DBMS_LDAP.BERVAL_COLLECTION

);
```

Parameters

Following table describes populate_mod_array procedure parameters:

Table 9-64   POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.

Table 9-64   (Cont.) POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters

Parameter	Description
mod_op	This field specifies the type of modification to perform, MOD_ADD, MOD_DELETE or MOD_REPLACE.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the attribute values to add, delete, or replace. It is for the binary values.

Exceptions

Following table describes populate_mod_array procedure exceptions:

Table 9-65   POPULATE_MOD_ARRAY (Binary Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is invoked after DBMS_LDAP.create_mod_array is called.

Related Functions

DBMS_LDAP.create_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

9.4.23 PROCEDURE populate_mod_array (Binary Version. Uses BLOB Data Type)

Populates one set of attribute information for add or modify operations. This procedure call occurs after DBMS_LDAP.create_mod_array() is called.

Syntax

```
PROCEDURE populate_mod_array
(
modptr IN DBMS_LDAP.MOD_ARRAY,
mod_op IN PLS_INTEGER,
mod_type IN VARCHAR2,
```

```
modbval IN DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

The following table lists populate_mod_array parameters:

Table 9-66   POPULATE_MOD_ARRAY (Binary) Parameters

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform, MOD_ADD, MOD_DELETE or MOD_REPLACE.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the binary attribute values to add, delete, or replace.

Exceptions

The following table lists populate_mod_array parameters:

Table 9-67   POPULATE_MOD_ARRAY (Binary) Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is invoked after DBMS_LDAP.create_mod_array is called.

Related Functions

DBMS_LDAP.create_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

9.4.24 FUNCTION get_values_blob

The function `get_values_blob()` can be used to retrieve larger values of attributes that have a binary syntax.

Syntax

```
FUNCTION get_values_blob
(
ld IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2
)
RETURN BLOB_COLLECTION;
```

Parameters

The following table lists get_values_blob parameters:

Table 9-68   GET_VALUES_BLOB Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.
attr	The string name of the attribute for which values are being sought.

Return Values

The following table lists get_values_blob return values:

Table 9-69   get_values_blob Return Values

Value	Description
BLOB_COLLECTION	A PL/SQL BLOB collection containing all the values of the given attribute.
NULL	No values are associated with the given attribute.

Exceptions

Following table lists get_values_blob exceptions:

Table 9-70   get_values_blob Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.

Table 9-70   (Cont.) get_values_blob Exceptions

Exception	Description
`invalid message`	Raised if the incoming entry handle is invalid.

Usage Notes

The function `get_values_blob()` can only be called after the handle to an entry has been retrieved by a call to either `first_entry()` or `next_entry()`. The name of the attribute may be known beforehand or can also be determined by a call to `first_attribute()` or `next_attribute()`. This function can be used to retrieve both binary and nonbinary attribute values.

Related Functions

`DBMS_LDAP.first_entry()`, `DBMS_LDAP.next_entry()`, `DBMS_LDAP.count_values_blob()`, `DBMS_LDAP.get_values()`.

9.4.25 FUNCTION count_values_blob

Counts the number of values returned by `DBMS_LDAP.get_values_blob()`.

Parameters

Return Values

Exceptions

See Also

Syntax

```
FUNCTION count_values_blob
(
values IN DBMS_LDAP.BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists count_values_blob parameters:

Table 9-71   COUNT_VALUES_BLOB Parameters

Parameter	Description
`values`	The collection of large binary values.

Return Values

Following table lists count_values_blob return values:

Table 9-72   COUNT_VALUES_BLOB Return Values

Values	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

The function count_values_blob() raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.count_values(), DBMS_LDAP.get_values_blob().

9.4.26 FUNCTION value_free_blob

Frees the memory associated with BLOB_COLLECTION returned by
DBMS_LDAP.get_values_blob().

Parameters

Exceptions

See Also

Syntax

```
PROCEDURE value_free_blob
(
vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

Following table lists value_free_blob parameters:

Table 9-73   VALUE_FREE_BLOB Parameters

Parameter	Description
vals	The collection of large binary values returned by DBMS_LDAP.get_values_blob().

Return Values

None

Exceptions

value_free_blob() raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.get_values_blob().

9.4.27 FUNCTION modify_s

Performs a synchronous modification of an existing LDAP directory entry.

Parameters

Return Values

Exceptions

Usage Notes

See Also

Syntax

```
FUNCTION modify_s
(

ld      IN DBMS_LDAP.SESSION,
entrydn IN VARCHAR2,
modptr  IN DBMS_LDAP.MOD_ARRAY

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists modify_s function parameters:

Table 9-74   MODIFY_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to DBMS_LDAP.init().
entrydn	This parameter specifies the name of the directory entry whose contents are to be modified.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array().

Return Values

Following table lists modify_s function return values:

Table 9-75   MODIFY_S Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Following table lists modify_s function exceptions:

Table 9-76   MODIFY_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

This function call has to follow successful calls of `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

Related Functions

`DBMS_LDAP.create_mod_array()`, `DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

9.4.28 FUNCTION add_s

Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()` must be called.

Syntax

```
FUNCTION add_s
(

ld      IN DBMS_LDAP.SESSION,
entrydn IN VARCHAR2,
modptr  IN DBMS_LDAP.MOD_ARRAY

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists add_s function parameters:

Table 9-77   ADD_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init().
entrydn	This parameter specifies the name of the directory entry to be created.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array().

Return Values

Following table lists add_s function return values:

Table 9-78   ADD_S Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Following table lists add_s function exceptions:

Table 9-79   ADD_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

The parent entry of the entry to be added must already exist in the directory. This function call has to follow successful calls to DBMS_LDAP.create_mod_array() and DBMS_LDAP.populate_mod_array().

Related Functions

DBMS_LDAP.create_mod_array(), DBMS_LDAP.populate_mod_array(), DBMS_LDAP.modify_s(), and DBMS_LDAP.free_mod_array().

9.4.29 PROCEDURE free_mod_array

Frees the memory allocated by DBMS_LDAP.create_mod_array().

Syntax

```
PROCEDURE free_mod_array
(

modptr IN DBMS_LDAP.MOD_ARRAY

);
```

Parameters

Following table lists free_mod_array procedure parameters:

Table 9-80   FREE_MOD_ARRAY Procedure Parameters

Parameter	Description
modptr	This parameter is the handle to an LDAP mod structure returned by a successful call to DBMS_LDAP.create_mod_array().

Return Values

None

Exceptions

free_mod_array raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.populate_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.create_mod_array().

9.4.30 FUNCTION count_values

Counts the number of values returned by DBMS_LDAP.get_values().

Syntax

```
FUNCTION count_values
(

values IN DBMS_LDAP.STRING_COLLECTION

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists count_values function parameters:

Table 9-81   COUNT_VALUES Function Parameters

Parameter	Description
values	The collection of string values.

Return Values

Following table lists count_values function returrn values:

Table 9-82   COUNT_VALUES Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

count_values raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.count_values_len(), DBMS_LDAP.get_values().

9.4.31 FUNCTION count_values_len

Counts the number of values returned by DBMS_LDAP.get_values_len().

Syntax

```
FUNCTION count_values_len
(
values IN DBMS_LDAP.BINVAL_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists count_values_len function parameters:

Table 9-83   COUNT_VALUES_LEN Function Parameters

Parameter	Description
values	The collection of binary values.

Return Values

Following table lists count_values_len return values:

Table 9-84   COUNT_VALUES_LEN Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

count_values_len raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

9.4.32 FUNCTION rename_s

Renames an LDAP entry synchronously.

Syntax

```
FUNCTION rename_s
(

ld           IN SESSION,
dn           IN VARCHAR2,
newrdn       IN VARCHAR2,
newparent    IN VARCHAR2,
deleteoldrdn IN PLS_INTEGER,
serverctrls  IN LDAPCONTROL,
clientctrls  IN LDAPCONTROL

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists rename_s function parameters:

Table 9-85   RENAME_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to DBMS_LDAP.init().
dn	This parameter specifies the name of the directory entry to be renamed or moved.
newrdn	This parameter specifies the new RDN.
newparent	This parameter specifies the DN of the new parent.
deleteoldrdn	This parameter specifies whether the old RDN should be retained. If this value is 1, the old RDN is removed.
serverctrls	Currently not supported.
clientctrls	Currently not supported.

Return Values

Following table lists rename_s function return values:

Table 9-86   RENAME_S Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Following table lists rename_s function exceptions:

Table 9-87   RENAME_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_entry_dn	Invalid LDAP DN.
invalid_rdn	Invalid LDAP RDN.
invalid_newparent	Invalid LDAP newparent.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.

Usage Notes

None

Related Functions

`DBMS_LDAP.modrdn2_s().`

9.4.33 FUNCTION explode_dn

Breaks a DN up into its components.

Syntax

```
FUNCTION explode_dn
(

dn      IN VARCHAR2,
notypes IN PLS_INTEGER

)

RETURN STRING_COLLECTION;
```

Parameters

Following table lists explode_dn function parameters:

Table 9-88   EXPLODE_DN Function Parameters

Parameter	Description
dn	This parameter specifies the name of the directory entry to be broken up.
notypes	This parameter specifies whether the attribute tags are returned. If this value is not `0`, no attribute tags are returned.

Return Values

Following table lists explode_dn function return values:

Table 9-89   EXPLODE_DN Function Return Values

Value	Description
STRING_COLLECTION	An array of strings. If the DN cannot be broken up, `NULL` is returned.

Exceptions

Following table lists explode_dn function exceptions:

Table 9-90   EXPLODE_DN Function Exceptions

Exception	Description
invalid_entry_dn	Invalid LDAP DN.
invalid_notypes	Invalid LDAP notypes value.

Usage Notes

None

Related Functions

DBMS_LDAP.get_dn().

9.4.34 FUNCTION open_ssl

Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Syntax

```
FUNCTION open_ssl
(

ld              IN SESSION,
sslwrl          IN VARCHAR2,
sslwalletpasswd IN VARCHAR2,
sslauth         IN PLS_INTEGER

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists open_ssl function parameters:

Table 9-91   OPEN_SSL Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session that is returned by a successful call to DBMS_LDAP.init().
sslwrl	This parameter specifies the wallet location. The format is file:path. Required for one-way or two-way SSL connections.
sslwalletpasswd	This parameter specifies the wallet password. Required for one-way or two-way SSL connections.
sslauth	This parameter specifies the SSL Authentication Mode. (1 for no authentication, 2 for one-way authentication required, 3 for two-way authentication).

Return Values

Following table lists open_ssl function return values:

Table 9-92   OPEN_SSL Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

Following table lists open_ssl exceptions:

Table 9-93   OPEN_SSL Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_ssl_wallet_loc	Invalid LDAP SSL wallet location.
invalid_ssl_wallet_passw d	Invalid LDAP SSL wallet password.
invalid_ssl_auth_mode	Invalid LDAP SSL authentication mode.

Usage Notes

Need to call DBMS_LDAP.init() first to acquire a valid ldap session.

Related Functions

DBMS_LDAP.init().

9.4.35 FUNCTION msgfree

This function frees the chain of messages associated with the message handle returned by synchronous search functions.

Syntax

```
FUNCTION msgfree
(

res             IN MESSAGE

)

RETURN PLS_INTEGER;
```

Parameters

Following table lists msgfree function parameters:

Table 9-94   MSGFREE Function Parameters

Parameter	Description
res	The message handle obtained by a call to one of the synchronous search routines.

Return Values

Following table lists msgfree function return values:

Table 9-95   MSGFREE Return Values

Value	Description
PLS_INTEGER	Indicates the type of the last message in the chain.
	The function might return any of the following values:
	• DBMS_LDAP.LDAP_RES_BIND
	• DBMS_LDAP.LDAP_RES_SEARCH_ENTRY
	• DBMS_LDAP.LDAP_RES_SEARCH_REFERENCE
	• DBMS_LDAP.LDAP_RES_SEARCH_RESULT
	• DBMS_LDAP.LDAP_RES_MODIFY
	• DBMS_LDAP.LDAP_RES_ADD
	• DBMS_LDAP.LDAP_RES_DELETE
	• DBMS_LDAP.LDAP_RES_MODDN
	• DBMS_LDAP.LDAP_RES_COMPARE
	• DBMS_LDAP.LDAP_RES_EXTENDED

Exceptions

msgfree raises no exceptions.

Usage Notes

None

Related Functions

DBMS_LDAP.search_s(), DBMS_LDAP.search_st().

9.4.36 FUNCTION ber_free

This function frees the memory associated with a handle to BER ELEMENT.

Syntax

```
FUNCTION ber_free
(
```

```
ber_elem IN BER_ELEMENT,
freebuf  IN PLS_INTEGER

)
```

Parameters

Following table lists ber_free function parameters:

Table 9-96   BER_FREE Function Parameters

Parameter	Description
`ber_elem`	A handle to `BER ELEMENT`.
`freebuf`	The value of this flag should be `0` while the `BER ELEMENT` returned from `DBMS_LDAP.first_attribute()` is being freed. For any other case, the value of this flag should be `1`.
	The default value of this parameter is zero.

Return Values

`ber_free` returns no values.

Exceptions

`ber_free` raises no exceptions.

Usage Notes

None

Related Functions

`DBMS_LDAP.first_attribute()`,`DBMS_LDAP.next_attribute()`.

9.4.37 FUNCTION nls_convert_to_utf8

The `nls_convert_to_utf8()` function converts the input string containing database character set data to UTF-8 character set data and returns it.

Syntax

```
Function nls_convert_to_utf8
(
    data_local IN VARCHAR2
)
    RETURN VARCHAR2;
```

Parameters

Following table lists nls_convert_to_utf8 function parameters:

Table 9-97   Parameters for nls_convert_to_utf8

Parameter	Description
data_local	Contains the database character set data.

Return Values

Following table lists nlss_convert_to_utf8 parameters:

Table 9-98   Return Values for nls_convert_to_utf8

Value	Description
VARCHAR2	UTF-8 character set data string.

Exceptions

None

Usage Notes

The functions in DBMS_LDAP package expect the input data to be UTF-8 character set data if the UTF8_CONVERSION package variable is set to FALSE. The nls_convert_to_utf8() function converts database character set data to UTF-8 character set data.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package expect input data to be database character set data.

Related Functions

DBMS_LDAP.nls_convert_from_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

9.4.38 FUNCTION nls_convert_from_utf8

The nls_convert_from_utf8() function converts the input string collection containing UTF-8 character set data to database character set data. It then returns this data.

Syntax

```
Function nls_convert_from_utf8
(
    data_utf8 IN STRING_COLLECTION
)
    RETURN STRING_COLLECTION;
```

Parameters

Following table lists nls_convert_from_utf8 parameters:

Table 9-99   Parameter for nls_convert_from_utf8

Parameter	Description
data_utf8	Collection of strings containing UTF-8 character set data.

Return Values

Following table lists nls_convert_from_utf8 parameters:

Table 9-100   Return Value for nls_convert_from_utf8

Value	Description
VARCHAR2	Collection of strings containing database character set data.

Exceptions

The functions in the DBMS_LDAP package return UTF-8 character set data if the UTF8_CONVERSION package variable is set to FALSE. nls_convert_from_utf8() converts the output data from the UTF-8 character set to the database character set. If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package return database character set data.

Usage Notes

None

Related Functions

DBMS_LDAP.nls_convert_to_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

9.4.39 FUNCTION nls_get_dbcharset_name

The nls_get_dbcharset_name() function returns a string containing the database character set name.

Syntax

```
Function nls_get_dbcharset_name

    RETURN VARCHAR2;
```

Parameters

None.

Return Values

Following table lists return value for nls_get_dbcharset_name:

Table 9-101   Return Value for nls_get_dbcharset_name

Value	Description
VARCHAR2	String containing the database character set name.

Exceptions

None

Usage Notes

None

Related Functions

DBMS_LDAP.nls_convert_to_utf8(), DBMS_LDAP.nls_convert_from_utf8().

10
Java API Reference

The standard Java APIs for Oracle Internet Directory are available as the Java Naming and Directory Interface (JNDI).
For reference information about the JNDI, see:

`http://www.oracle.com/technetwork/java/jndi/index.html`.

For the Oracle extensions to the standard APIs, Refer to *Oracle Fusion Middleware Java API Reference for Oracle Internet Directory*.

11

DBMS_LDAP_UTL PL/SQL Reference

The `DBMS_LDAP_UTL` package contains Oracle Extension utility functions for user, group and subscribers.

- Subprograms at a Glance
- Handling Subprograms
- Function Return Code Summary
- Data Type Summary

11.1 Subprograms at a Glance

Subprograms are used to retrieve information from users, groups and subscribers.

The following topics provide the summary of subprograms at various levels:

- dbms_ldap_utl User-Related Subprograms
- dbms_ldap_utl Group-Related Subprograms
- dbms_ldap_utl Subscriber-Related Subprograms
- dbms_ldap_utl Miscellaneous Subprograms

11.1.1 dbms_ldap_utl User-Related Subprograms

`dbms_ldap_utl` user-related subprograms are used to authenticate, retrieve and modify the user properties from an LDAP server.

Table 11-1 describes `dbms_ldap_utl` user-related subprograms.

Table 11-1   DBMS_LDAP_UTL User-Related Subprograms

Function or Procedure	Purpose
Function authenticate_user	Authenticates a user against an LDAP server.
Function create_user_handle	Creates a user handle.
Function set_user_handle_properties	Associates the given properties to the user handle.
Function get_user_properties	Retrieves user properties from an LDAP server.
Function set_user_properties	Modifies the properties of a user.
Function get_user_extended_properties	Retrieves user extended properties.
Function get_user_dn	Retrieves a user DN.
Function check_group_membership	Checks whether a user is member of a given group.
Function locate_subscriber_for_user	Retrieves the subscriber for the given user.
Function get_group_membership	Retrieves a list of groups of which the user is a member.

11.1.2 dbms_ldap_utl Group-Related Subprograms

dbms_ldap_utl group-related subprograms are used to create and retrieve information from an LDAP server.

Table 11-2 describes dbms_ldap_utl group-related subprograms.

Table 11-2   DBMS_LDAP_UTL Group-Related Subprograms

Function or Procedure	Purpose
Function create_group_handle	Creates a group handle.
Function set_group_handle_properties	Associates the given properties with the group handle.
Function get_group_properties	Retrieves group properties from an LDAP server.
Function get_group_dn	Retrieves a group DN.

11.1.3 dbms_ldap_utl Subscriber-Related Subprograms

dbms_ldap_utl subscriber-related subprograms are used to create and retrieve subscriber properties from an LDAP server.

Table 11-3 describes dbms_ldap_utl subscriber-related subprograms.

Table 11-3   DBMS_LDAP_UTL Subscriber-Related Subprograms

Function or Procedure	Purpose
Function create_subscriber_handle	Creates a subscriber handle.
Function get_subscriber_properties	Retrieves subscriber properties from an LDAP server.
Function get_subscriber_dn	Retrieves a subscriber DN.

11.1.4 dbms_ldap_utl Miscellaneous Subprograms

dbms_ldap_utl miscellaneous subprograms are used to retrieve list of property names and binary values for a property etc.

Table 11-4 describes dbms_ldap_utl miscellaneous subprograms.

Table 11-4   DBMS_LDAP_UTL Miscellaneous Subprograms

Function or Procedure	Purpose
Function normalize_dn_with_case	Normalizes the DN string.
Function get_property_names	Retrieves a list of property names in a PROPERTY_SET.
Function get_property_values	Retrieves a list of values for a property name.
Function get_property_values_blob	Retrieves a list of large binary values for a property name.

Table 11-4   (Cont.) DBMS_LDAP_UTL Miscellaneous Subprograms

Function or Procedure	Purpose
Procedure property_value_free_blob	Frees the memory associated with `BLOB_COLLECTION` returned by `DBMS_LDAP_UTL.get_property_values_blob()`.
Function get_property_values_len	Retrieves a list of binary values for a property name.
Procedure free_propertyset_collection	Frees `PROPERTY_SET_COLLECTION`.
Function create_mod_propertyset	Creates a `MOD_PROPERTY_SET`.
Function populate_mod_propertyset	Populates a `MOD_PROPERTY_SET` structure.
Procedure free_mod_propertyset	Frees a `MOD_PROPERTY_SET`.
Procedure free_handle	Frees handles.
Function check_interface_version	Checks for support of the interface version.

11.2 Handling Subprograms

`DBMS_LDAP_UTL` package contains user, group, property and subscriber related subprograms.

The following topics describe the various subprograms available for the `DBMS_LDAP_UTL` package:

- Using User-Related Subprograms
- Using Group-Related Subprograms
- Using Subscriber-Related Subprograms
- Using Property-Related Subprograms

11.2.1 Using User-Related Subprograms

A user is represented by the `DBMS_LDAP_UTL.HANDLE` data type. You can create a user handle by using a DN, GUID, or simple name, along with the appropriate subscriber handle. When a simple name is used, additional information from the root Oracle Context and the subscriber Oracle Context is used to identify the user.

The following example shows a user handle being created:

```
retval := DBMS_LDAP_UTL.create_user_handle(
user_handle,
DBMS_LDAP_UTL.TYPE_DN,
"cn=user1,cn=users,o=example,dc=com"
);
```

This user handle must be associated with an appropriate subscriber handle. If, for example, `subscriber_handle` is `o=example,dc=com`, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_user_handle_properties(
user_handle,
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
subscriber_handle
);
```

Common uses of user handles include setting and getting user properties and authenticating the user. Here is a handle that authenticates a user:

```
retval := DBMS_LDAP_UTL.authenticate_user(
my_session
user_handle
DBMS_LDAP_UTL.AUTH_SIMPLE,
"welcome"
NULL
);
```

In this example, the user is authenticated using a clear text password `welcome`.

Here is a handle that retrieves a user's telephone number:

```
--my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) :='telephonenumber';
retval := DBMS_LDAP_UTL.get_user_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

11.2.2 Using Group-Related Subprograms

A group is represented using by using the `DBMS_LDAP_UTL.HANDLE` data type. A group handle represents a valid group entry. You can create a group handle by using a DN, GUID or a simple name, along with the appropriate subscriber handle. When a simple name is used, additional information from the Root Oracle Context and the Subscriber Oracle Context is used to identify the group.

Here is an example of a group handle creation:

```
retval := DBMS_LDAP_UTL.create_group_handle(
group_handle,
DBMS_LDAP_UTL.TYPE_DN,
"cn=group1,cn=Groups,o=example,dc=com"
);
```

This group handle has to be associated with an appropriate subscriber handle. For example, given a subscriber handle: *subscriber_handle* representing `o=example,dc=com`, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_group_handle_properties(
group_handle,
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
subscriber_handle
);
```

A sample use of group handle is getting group properties. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) :='uniquemember';
retval := DBMS_LDAP_UTL.get_group_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

ORACLE®

The group-related subprograms also support membership-related functionality. Given a user handle, you can find out if it is a direct or a nested member of a group by using the `DBMS_LDAP_UTL.check_group_membership()` function. Here is an example:

```
retval := DBMS_LDAP_UTL.check_group_membership(
session,
user_handle,
group_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP
```

You can also obtain a list of groups that a particular group belongs to, using the `DBMS_LDAP_UTL.get_group_membership()` function. For example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) :='cn';
retval := DBMS_LDAP_UTL.get_group_membership(
my_session,
user_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,
my_attrs
my_pset_coll
);
```

11.2.3 Using Subscriber-Related Subprograms

A subscriber is represented by using `dbms_ldap_utl.handle` data type. You can create a subscriber handle by using a DN, GUID or simple name. When a simple name is used, additional information from the root Oracle Context is used to identify the subscriber.

This example shows a subscriber handle being created:

```
retval := DBMS_LDAP_UTL.create_subscriber_handle(

subscriber_handle,
DBMS_LDAP_UTL.TYPE_DN,
"o=example,dc=com"
);
```

`subscriber_handle` is created by it's DN: `o=oracle,dc=com`.

Getting subscriber properties is one common use of a subscriber handle. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
      my_attrs(1) :='orclguid';
      retval := DBMS_LDAP_UTL.get_subscriber_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

11.2.4 Using Property-Related Subprograms

Many of the user-related, subscriber-related, and group-related subprograms return `DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION`, which is a collection of one or more LDAP entries representing results. Each of these entries is represented by a

DBMS_LDAP_UTL.PROPERTY_SET. A PROPERTY_SET may contain attributes—that is, properties—and its values.

Here is an example that illustrates the retrieval of properties from DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) :='cn';

retval := DBMS_LDAP_UTL.get_group_membership(
my_session,
user_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,
my_attrs,
my_pset_coll
);

IF my_pset_coll.count > 0 THEN
      FOR i in my_pset_coll.first .. my_pset_coll.last LOOP
--    my_property_names is of type DBMS_LDAP.STRING_COLLECTION
       retval := DBMS_LDAP_UTL.get_property_names(
pset_coll(i),
property_names
      IF my_property_names.count > 0 THEN
          FOR j in my_property_names.first .. my_property_names.last LOOP
           retval := DBMS_LDAP_UTL.get_property_values(
pset_coll(i),
property_names(j),
property_values
           if my_property_values.COUNT > 0 then
            FOR k in my_property_values.FIRST..my_property_values.LAST LOOP
                 DBMS_OUTPUT.PUT_LINE(my_property_names(j)  || ':'
                    ||my_property_values(k));
                END LOOP; -- For each value
           else
              DBMS_OUTPUT.PUT_LINE('NO VALUES FOR' || my_property_names(j));
           end if;
         END LOOP; -- For each property name
        END IF; -- IF my_property_names.count > 0
     END LOOP; -- For each propertyset
  END IF; -- If my_pset_coll.count > 0
```

use_handle is a user handle. my_pset_coll contains all the nested groups that user_handle belongs to. The code loops through the resulting entries and prints out the cn of each entry.

11.3 Function Return Code Summary

The DBMS_LDAP_UTL package includes functions which return error codes.

Table 11-5   Function Return Codes

Name	Return Code	Description
SUCCESS	0	Operation successful.

Table 11-5   (Cont.) Function Return Codes

Name	Return Code	Description
GENERAL_ERROR	-1	This error code is returned on failure conditions other than those conditions listed here.
PARAM_ERROR	-2	Returned by all functions when an invalid input parameter is encountered.
NO_GROUP_MEMBERSHIP	-3	Returned by user-related functions and group functions when the user is not a member of a group.
NO_SUCH_SUBSCRIBER	-4	Returned by subscriber-related functions when the subscriber does not exist in the directory.
NO_SUCH_USER	-5	Returned by user-related functions when the user does not exist in the directory.
NO_ROOT_ORCL_CTX	-6	Returned by most functions when the root oracle context does not exist in the directory.
MULTIPLE_SUBSCRIBER_ENTRIES	-7	Returned by subscriber-related functions when multiple subscriber entries are found for the given subscriber nickname.
INVALID_ROOT_ORCL_CTX	-8	Root Oracle Context does not contain all the required information needed by the function.
NO_SUBSCRIBER_ORCL_CTX	-9	Oracle Context does not exist for the subscriber.
INVALID_SUBSCRIBER_ORCL_CTX	-10	Oracle Context for the subscriber is invalid.
MULTIPLE_USER_ENTRIES	-11	Returned by user-related functions when multiple user entries exist for the given user nickname.
NO_SUCH_GROUP	-12	Returned by group related functions when a group does not exist in the directory.
MULTIPLE_GROUP_ENTRIES	-13	Multiple group entries exist for the given group nickname in the directory.
ACCT_TOTALLY_LOCKED_EXCEPTION	-14	Returned by DBMS_LDAP_UTL.authenticate_user() function when a user account is locked. This error is based on the password policy set in the subscriber oracle context.
AUTH_PASSWD_CHANGE_WARN	-15	This return code is deprecated.
AUTH_FAILURE_EXCEPTION	-16	Returned by DBMS_LDAP_UTL.authenticate_user() function when user authentication fails.
PWD_EXPIRED_EXCEPTION	-17	Returned by DBMS_LDAP_UTL.authenticate_user() function when the user password has expired. This is a password policy error.

ORACLE®

Table 11-5   (Cont.) Function Return Codes

Name	Return Code	Description
RESET_HANDLE	-18	Returned when entity handle properties are being reset by the caller.
SUBSCRIBER_NOT_FOUND	-19	Returned by `DBMS_LDAP-UTL.locate_subscriber_for_user()` function when it is unable to locate the subscriber.
PWD_EXPIRE_WARN	-20	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when the user password is about to expire. This is a password policy error.
PWD_MINLENGTH_ERROR	-21	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password is less than the minimum required length. This is a password policy error.
PWD_NUMERIC_ERROR	-22	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password does not contain at least one numeric character. This is a password policy error.
PWD_NULL_ERROR	-23	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password is an empty password. This is a password policy error.
PWD_INHISTORY_ERROR	-24	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password is the same as the previous password. This is a password policy error.
PWD_ILLEGALVALUE_ERROR	-25	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password has an illegal character. This is a password policy error.
PWD_GRACELOGIN_WARN	-26	Returned by `DBMS_LDAP_UTL.authenticate_user()` function to indicate that the user password has expired and the user has been given a grace login. This is a password policy error.
PWD_MUSTCHANGE_ERROR	-27	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when user password must be changed. This is a password policy error.

Table 11-5   (Cont.) Function Return Codes

Name	Return Code	Description
`USER_ACCT_DISABLED_ERROR`	-29	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when user account has been disabled. This is a password policy error.
`PROPERTY_NOT_FOUND`	-30	Returned by user-related functions while searching for a user property in the directory.

11.4 Data Type Summary

The `DBMS_LDAP_UTL` package includes the data types to hold entity and properties of an entity.

The `DBMS_LDAP_UTL` package uses the data types in the following table:

Table 11-6   DBMS_LDAP_UTL Data Types

Data Type	Purpose
`HANDLE`	Used to hold the entity.
`PROPERTY_SET`	Used to hold the properties of an entity.
`PROPERTY_SET_COLLECTION`	List of `PROPERTY_SET` structures.
`MOD_PROPERTY_SET`	Structure to hold modify operations on an entity.

11.5 User-Related Subprograms

A user is represented by the `DBMS_LDAP_UTL.HANDLE` data type. You can create a user handle by using a DN, GUID, or simple name, along with the appropriate subscriber handle.

When a simple name is used, additional information from the root Oracle Context and the subscriber Oracle Context is used to identify the user.

The following topics describe the user related subprograms:

- Function authenticate_user
- Function create_user_handle
- Function set_user_handle_properties
- Function get_user_properties
- Function set_user_properties
- Function get_user_extended_properties
- Function get_user_dn

11.5.1 Function authenticate_user

The function `authenticate_user()` authenticates the user against Oracle Internet Directory.

Syntax

```
FUNCTION authenticate_user
(
ld IN SESSION,
user_handle IN HANDLE,
auth_type IN PLS_INTEGER,
credentials IN VARCHAR2,
binary_credentials IN RAW
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists authenticate_user function parameters:

Table 11-7   authenticate_user Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
auth_type	PLS_INTEGER	Type of authentication. The only valid value is DBMS_LDAP_UTL.AUTH_SIMPLE
credentials	VARCHAR2	The user credentials.
binary_credentials	RAW	The binary credentials. This parameter is optional. It can be NULL by default.

Return Values

Following table lists authenticate_user function return values:

Table 11-8   authenticate_user Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.

Table 11-8   (Cont.) authenticate_user Function Return Values

Value	Description
`DBMS_LDAP_UTL.GENERAL_ERROR`	Authentication failed.
`DBMS_LDAP_UTL.NO_SUCH_USER`	User does not exist.
`DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES`	The user has multiple DN entries.
`DBMS_LDAP_UTL.INVALID_SUBSCRIBER_ORCL_CTX`	Invalid Subscriber Oracle Context.
`DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER`	Subscriber doesn't exist.
`DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES`	The subscriber has multiple DN entries.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid Root Oracle Context.
`DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP`	User account is locked.
`DBMS_LDAP_UTL.AUTH_PASSWD_CHANGE_WARN`	This return value is deprecated.
`DBMS_LDAP_UTL.AUTH_FAILURE_EXCP`	Authentication failed.
`DBMS_LDAP_UTL.PWD_EXPIRED_EXCP`	User password has expired.
`DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN`	Grace login for user.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures that occurred when LDAP operations were carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

11.5.2 Function create_user_handle

The function `create_user_handle()` creates a user handle.

Syntax

```
FUNCTION create_user_handle
(
user_hd OUT HANDLE,
user_type IN PLS_INTEGER,
user_id IN VARCHAR2,
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists create_user_handle function parameters:

Table 11-9   CREATE_USER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_hd	HANDLE	A pointer to a handle to a user.
user_type	PLS_INTEGER	The type of user ID that is passed. Valid values for this argument are as follows: • DBMS_LDAP_UTL.TYPE_DN • DBMS_LDAP_UTL.TYPE_GUID • DBMS_LDAP_UTL.TYPE_NICKNAME
user_id	VARCHAR2	The user ID representing the user entry.

Return Values

Following table lists create_user_handle function parameters:

Table 11-10   CREATE_USER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.set_user_handle_properties().

11.5.3 Function set_user_handle_properties

The function set_user_handle_properties() configures the user handle properties.

Syntax

```
FUNCTION set_user_handle_properties
(
user_hd IN HANDLE,
property_type IN PLS_INTEGER,
property IN HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists the parameters:

Table 11-11   SET_USER_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_hd	HANDLE	A pointer to a handle to a user.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: - DBMS_LDAP_UTL.SUBSCRIBER_HANDLE.
property	HANDLE	The property describing the user entry.

Return Values

Following table lists the set_user_handle_properties function return values:

Table 11-12   SET_USER_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

ORACLE

Usage Notes

The subscriber handle does not have to be set in User Handle Properties if the user handle is created with `TYPE_DN` or `TYPE_GUID` as the user type.

Related Functions

`DBMS_LDAP_UTL.get_user_properties().`

11.5.4 Function get_user_properties

The function `get_user_properties()` retrieves the user properties.

Syntax

```
FUNCTION get_user_properties
(
ld IN SESSION,
user_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists get_user_properties function parameters:

Table 11-13   GET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`attrs`	`STRING_COLLECTION`	The list of user attributes to retrieve.
`ptype`	`PLS_INTEGER`	Type of properties to return. These are valid values: • `DBMS_LDAP_UTL.ENTRY_PROPERTIES` • `DBMS_LDAP_UTL.NICKNAME_PROPERTY`
`ret-pset_collection`	`PROPERTY_SET_COLLECTION`	User details contained in attributes requested by the caller.

Return Values

Following table lists get_user_properties function return values:

Table 11-14   GET_USER_PROPERTIES Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.NO_SUCH_USER`	User does not exist.
`DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES`	The user has multiple DN entries.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid root Oracle Context.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function requires the following:

- A valid LDAP session handle, which must be obtained from the `DBMS_LDAP.init()` function.
- A valid subscriber handle to be set in the group handle properties if the user type is of `DBMS_LDAP_UTL.TYPE_NICKNAME`.

This function does not identify a `NULL` subscriber handle as a default subscriber. The default subscriber can be obtained from `DBMS_LDAP_UTL.create_subscriber_handle()`, where a `NULL subscriber_id` is passed as an argument.

If the group type is either `DBMS_LDAP_UTL.TYPE_GUID` or `DBMS_LDAP_UTL.TYPE_DN`, the subscriber handle need not be set in the user handle properties. If the subscriber handle is set, it is ignored.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

11.5.5 Function set_user_properties

The function `set_user_properties()` modifies the properties of a user.

Syntax

```
FUNCTION set_user_properties
(
ld IN SESSION,
user_handle IN HANDLE,
pset_type IN PLS_INTEGER,
mod_pset IN PROPERTY_SET,
```

```
mod_op IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists the set_user_properties function parameters:

Table 11-15   SET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
pset_type	PLS_INTEGER	The type of property set being modified. A valid value is ENTRY_PROPERTIES.
mod_pset	PROPERTY_SET	Data structure containing modify operations to perform on the property set.
mod_op	PLS_INTEGER	The type of modify operation to be performed on the property set. Here are valid values: • ADD_PROPERTYSET • MODIFY_PROPERTYSET • DELETE_PROPERTYSET

Return Values

Following table lists set_user_properties return values:

Table 11-16   SET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.PWD_MIN_LENGTH_ERROR	Password length is less than the minimum required length.
DBMS_LDAP_UTL.PWD_NUMERIC_ERROR	Password must contain numeric characters.
DBMS_LDAP_UTL.PWD_NULL_ERROR	Password cannot be NULL.
DBMS_LDAP_UTL.PWD_INHISTORY_ERROR	Password cannot be the same as the one that is being replaced.

Table 11-16   (Cont.) SET_USER_PROPERTIES Function Return Values

Value	Description
`DBMS_LDAP_UTL.PWD_ILLEGALVALUE_ERROR`	Password contains illegal characters.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.get_user_properties()`.

11.5.6 Function get_user_extended_properties

The function `get_user_extended_properties()` retrieves user extended properties.

Syntax

```
FUNCTION get_user_extended_properties
(
ld IN SESSION,
user_handle IN HANDLE,
attrs IN STRING_COLLECTION
ptype IN PLS_INTEGER,
filter IN VARCHAR2,
rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists the get_user_extended_properties function parameters:

Table 11-17   GET_USER_EXTENDED_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`attrs`	`STRING_COLLECTION`	A list of attributes to fetch for the user.

Table 11-17   (Cont.) GET_USER_EXTENDED_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ptype`	`PLS_INTEGER`	The type of properties to return. Here is a valid value: – `DBMS_LDAP_UTL.EXTPROPTYPE_RAD`
`filter`	`VARCHAR2`	An LDAP filter to further refine the user properties returned by the function.
`ret_pset_collection`	`PROPERTY_SET_COLLECTION`	The user details containing the attributes requested by the caller.

Return Values

Following table lists get_user_extended_properties function return values:

Table 11-18   GET_USER_EXTENDED_PROPERTIES Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.NO_SUCH_USER`	User does not exist.
`DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES`	The user has multiple DN entries.
`USER_PROPERTY_NOT_FOUND`	User extended property does not exist.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid root Oracle Context.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.get_user_properties()`.

11.5.7 Function get_user_dn

The `function get_user_dn()` returns the user DN.

Syntax

```
FUNCTION get_user_dn
(
ld IN SESSION,
user_handle IN HANDLE,
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists get_user_dn parameters:

Table 11-19   GET_USER_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
dn	VARCHAR2	The user DN.

Return Values

Following table lists get_user_dn function return values:

Table 11-20   GET_USER_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Authentication failed.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Table 11-20   (Cont.) GET_USER_DN Function Return Values

Value	Description
`DBMS_LDAP error codes`	Return proper `DBMS_LDAP` error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`.

11.5.8 Function check_group_membership

The function `check_group_membership()` checks whether the user belongs to a group.

Syntax

```
FUNCTION check_group_membership
(
ld IN SESSION,
user_handle IN HANDLE,
group_handle IN HANDLE,
nested IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists check_group_membership function parameters:

Table 11-21   CHECK_GROUP_MEMBERSHIP Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`group_handle`	`HANDLE`	The group handle.
`nested`	`PLS_INTEGER`	The type of membership the user holds in groups. Here are valid values: • `DBMS_LDAP_UTL.NESTED_MEMBERSHIP` • `DBMS_LDAP_UTL.DIRECT_MEMBERSHIP`

Return Values

The following table lists check_group_membership function return values:

Table 11-22   CHECK_GROUP_MEMBERSHIP Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	If user is a member.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_GROUP_MEMBERSHIP	If user is not a member.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to
DBMS_LDAP.init().

Related Functions

DBMS_LDAP.get_group_membership().

11.5.9 Function locate_subscriber_for_user

The function locate_subscriber_for_user() retrieves the subscriber for the given user and
returns a handle to it.

Syntax

```
FUNCTION locate_subscriber_for_user
(
ld IN SESSION,
user_handle IN HANDLE,
subscriber_handle OUT HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists locate_subscriber_for_user function parameters:

Table 11-23   LOCATE_SUBSCRIBER_FOR_USER Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
subscriber_handle	HANDLE	The subscriber handle.

Return Values

Following table lists locate subscriber for user function return values:

Table 11-24   LOCATE SUBSCRIBER FOR USER Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER`	Subscriber doesn't exist.
`DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES`	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
`DBMS_LDAP_UTL.NO_SUCH_USER`	User doesn't exist.
`DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES`	Multiple number of user DN entries exist in the directory for the given user.
`DBMS_LDAP_UTL.SUBSCRIBER_NOT_FOUND`	Unable to locate subscriber for the given user.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid Root Oracle Context.
`DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP`	User account is locked.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_user_handle()`.

11.5.10 Function get_group_membership

The function `get_group_membership()` returns the list of groups to which the user is a member.

Syntax

```
FUNCTION get_group_membership
(
user_handle IN HANDLE,
nested IN PLS_INTEGER,
attr_list IN STRING_COLLECTION,
ret_groups OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists get_group_membership function parameters:

Table 11-25   GET_GROUP_MEMBERSHIP Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`nested`	`PLS_INTEGER`	The type of membership the user holds in groups. Here are valid values: • `DBMS_LDAP_UTL.NESTED_MEMBERSHIP` • `DBMS_LDAP_UTL.DIRECT_MEMBERSHIP`
`attr_list`	`STRING_COLLECTION`	A list of attributes to be returned.
`ret_groups`	`PROPERTY_SET_COLLECTION`	A pointer to a pointer to an array of group entries.

Return Values

The following table lists get_group_membership function return values:

Table 11-26   GET_GROUP_MEMBERSHIP Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`.

11.6 Group-Related Subprograms

A group is represented using by using the `DBMS_LDAP_UTL.HANDLE` data type. A group handle represents a valid group entry. You can create a group handle by using a DN, GUID or a simple name, along with the appropriate subscriber handle.

When a simple name is used, additional information from the Root Oracle Context and the Subscriber Oracle Context is used to identify the group.

The following topics describe group_related subprograms:

- Function create_group_handle
- Function set_group_handle_properties
- Function get_group_properties
- Function get_group_dn

11.6.1 Function create_group_handle

The function `create_group_handle()` creates a group handle.

Syntax

```
FUNCTION create_group_handle
(
group_hd OUT HANDLE,
group_type IN PLS_INTEGER,
group_id IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table create_group_handle function parameters:

Table 11-27   CREATE_GROUP_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
`group_hd`	`HANDLE`	A pointer to a handle to a group.
`group_type`	`PLS_INTEGER`	The type of group ID that is passed. Valid values for this argument are as follows: • `DBMS_LDAP_UTL.TYPE_DN` • `DBMS_LDAP_UTL.TYPE_GUID` • `DBMS_LDAP_UTL.TYPE_NICKNAME`
`group_id`	`VARCHAR2`	The group ID representing the group entry.

Return Values

Following table lists create_group_handle function return values:

Table 11-28   CREATE_GROUP_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

None

Related Functions

```
DBMS_LDAP_UTL.get_group_properties(),
DBMS_LDAP_UTL.set_group_handle_properties().
```

11.6.2 Function set_group_handle_properties

The function `set_group_handle_properties()` configures the group handle properties.

Syntax

```
FUNCTION set_group_handle_properties
(
group_hd IN HANDLE,
property_type IN PLS_INTEGER,
property IN HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Following table lists set_group_handle_properties function parameters:

Table 11-29   SET_GROUP_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to the handle to the group.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: DBMS_LDAP_UTL.GROUP_HANDLE
property	HANDLE	The property describing the group entry.

Return Values

Following table lists set_group_handle_properties function return values:

Table 11-30   SET_GROUP_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

The subscriber handle doesn't need to be set in Group Handle Properties if the group handle is created with TYPE_DN or TYPE_GUID as the group type.

Related Functions

DBMS_LDAP_UTL.get_group_properties().

11.6.3 Function get_group_properties

The function get_group_properties() retrieves the group properties.

Syntax

```
FUNCTION get_group_properties
(
ld IN SESSION,
group_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_group_properties function parameters:

Table 11-31   GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
attrs	STRING_COLLECTION	A list of attributes that must be fetched for the group.

Table 11-31   (Cont.) GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ptype	PLS_INTEGER	The type of properties to be returned. The valid value is DBMS_LDAP_UTL.ENTRY_PROPERTIES
ret_pset_coll	PROPERTY_SET_COLLECTION	The group details containing the attributes requested by the caller.

Return Values

The following table lists get_group_properties function return values:

Table 11-32   GET_GROUP_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function requires the following:

• A valid LDAP session handle which must be obtained from the DBMS_LDAP.init() function.

• A valid subscriber handle to be set in the group handle properties if the group type is of: DBMS_LDAP_UTL.TYPE_NICKNAME.

This function does not identify a NULL subscriber handle as a default subscriber. The default subscriber can be obtained from DBMS_LDAP_UTL.create_subscriber_handle(), where a NULL subscriber_id is passed as an argument.

If the group type is either DBMS_LDAP_UTL.TYPE_GUID or DBMS_LDAP_UTL.TYPE_DN, the subscriber handle does not have to be set in the group handle properties. If the subscriber handle is set, it is ignored.

Related Functions

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_group_handle().

11.6.4 Function get_group_dn

The function get_group_dn() returns the group DN.

Syntax

```
FUNCTION get_group_dn
(
ld IN SESSION,
group_handle IN HANDLE
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_group_dn function parameters:

Table 11-33   GET_GROUP_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
dn	VARCHAR2	The group DN.

Return Values

The following table lists get_group_dn function return values:

Table 11-34   GET_GROUP_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.

Table 11-34   (Cont.) GET_GROUP_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that are encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

Related Functions

DBMS_LDAP.init().

11.7 Subscriber-Related Subprograms

A subscriber is represented by using dbms_ldap_utl.handle data type. You can create a subscriber handle by using a DN, GUID or simple name.

When a simple name is used, additional information from the root Oracle Context is used to identify the subscriber.

The Subscriber related subprograms are described below:

- Function create_subscriber_handle
- Function get_subscriber_properties
- Function get_subscriber_dn
- Function get_subscriber_ext_properties

11.7.1 Function create_subscriber_handle

The function create_subscriber_handle() creates a subscriber handle.

Syntax

```
FUNCTION create_subscriber_handle
(
subscriber_hd OUT HANDLE,
subscriber_type IN PLS_INTEGER,
subscriber_id IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists create_subscriber_handle function parameters:

Table 11-35   CREATE_SUBSCRIBER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
subscriber_hd	HANDLE	A pointer to a handle to a subscriber.
subscriber_type	PLS_INTEGER	The type of subscriber ID that is passed. Valid values for this argument are: • DBMS_LDAP_UTL.TYPE_DN • DBMS_LDAP_UTL.TYPE_GUID • DBMS_LDAP_UTL.TYPE_NICKNAME • DBMS_LDAP_UTL.TYPE_DEFAULT
subscriber_id	VARCHAR2	The subscriber ID representing the subscriber entry. This can be NULL if subscriber_type is DBMS_LDAP_UTL.TYPE_DEFAULT. In this case, the default subscriber is retrieved from the root Oracle Context.

Return Values

The following table lists create_subscriber_handle function return values:

Table 11-36   CREATE_SUBSCRIBER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_subscriber_properties().

11.7.2 Function get_subscriber_properties

The function get_subscriber_properties() retrieves the subscriber properties for the given subscriber handle.

Syntax

```
FUNCTION get_subscriber_properties
(
ld IN SESSION,
subscriber_handle IN HANDLE,
attrs IN STRING_COLLECTION,
```

```
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table get_subscriber_properties function parameters:

Table 11-37   GET_SUBSCRIBER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`subscriber_handle`	`HANDLE`	The subscriber handle.
`attrs`	`STRING_COLLECTION`	A list of attributes that must be retrieved for the subscriber.
`ptype`	`PLS_INTEGER`	Properties of the subscriber's Oracle Context to return. These are valid values: • `DBMS_LDAP_UTL.ENTRY_PROPERTIES` • `DBMS_LDAP_UTL.COMMON_PROPERTIES`
`ret_pset_coll`	`PROPERTY_SET_COLLECTION`	The subscriber details containing the attributes requested by the caller.

Return Values

The following table lists get_subscriber_properties function return values:

Table 11-38   GET_SUBSCRIBER_PROPERTIES Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER`	Subscriber doesn't exist.
`DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES`	Subscriber has a multiple number of DN entries.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid root Oracle Context.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures encountered while LDAP operations are carried out.

ORACLE®

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`, `DBMS_LDAP_UTL.create_subscriber_handle()`.

11.7.3 Function get_subscriber_dn

The function `get_subscriber_dn()` returns the subscriber DN.

Syntax

```
FUNCTION get_subscriber_dn
(
ld IN SESSION,
subscriber_handle IN HANDLE,
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_subscriber_dn function parameters:

Table 11-39   GET_SUBSCRIBER_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
dn	VARCHAR2	The subscriber DN.

Return Values

The following table lists get_subscriber_dn function return values:

Table 11-40   GET_SUBSCRIBER_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.

Table 11-40   (Cont.) GET_SUBSCRIBER_DN Function Return Values

Value	Description
`DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES`	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid root Oracle Context.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.
DBMS_LDAP error codes	Return proper `DBMS_LDAP` error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

Related Functions

`DBMS_LDAP.init()`.

11.7.4 Function get_subscriber_ext_properties

The function `get_subscriber_ext_properties()` retrieves the subscriber extended properties. Currently this can be used to retrieve the subscriber-wide default Resource Access Descriptors.

Syntax

```
FUNCTION get_subscriber_ext_properties
(
ld IN SESSION,
subscriber_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
filter IN VARCHAR2,
rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_subscriber_ext_properties function parameters:

Table 11-41   GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.

Table 11-41   (Cont.) GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
subscriber_handle	HANDLE	The subscriber handle.
attrs	STRING_COLLECTION	A list of subscriber attributes to retrieve.
ptype	PLS_INTEGER	The type of properties to return. A valid value is DBMS_LDAP_UTL.DEFAULT_RAD_PROPERTIES
filter	VARCHAR2	An LDAP filter to further refine the subscriber properties returned by the function.
ret_pset_collection	PROPERTY_SET_COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values

The following table lists get_user_extended_properties function return values:

Table 11-42   GET_USER_EXTENDED_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

Related Functions

DBMS_LDAP.init(), DBMS_LDAP_UTL.get_subscriber_properties().

11.8 Property-Related Subprograms

Many of the user-related, subscriber-related, and group-related subprograms return `DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION`, which is a collection of one or more LDAP entries representing results.

Each of these entries is represented by a `DBMS_LDAP_UTL.PROPERTY_SET`. A `PROPERTY_SET` may contain attributes—that is, properties—and its values.

11.9 Miscellaneous Subprograms

The miscellaneous subprograms in the `DBMS_LDAP_UTL` package perform a variety of different functions.

Miscellaneous subprograms are described as follows:

- Function normalize_dn_with_case
- Function get_property_names
- Function get_property_values
- Function get_property_values_len
- Procedure free_propertyset_collection
- Function create_mod_propertyset
- Function populate_mod_propertyset
- Procedure free_mod_propertyset
- Procedure free_handle
- Function check_interface_version
- Function get_property_values_blob
- Procedure property_value_free_blob

11.9.1 Function normalize_dn_with_case

The function `normalize_dn_with_case()` removes unnecessary white space characters from a DN and converts all characters to lowercase based on a flag.

Syntax

```
FUNCTION normalize_dn_with_case
(
dn IN VARCHAR2,
lower_case IN PLS_INTEGER,
norm_dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists normalize_dn_with_case function parameters:

Table 11-43   NORMALIZE_DN_WITH_CASE Function Parameters

Parameter Name	Parameter Type	Parameter Description
dn	VARCHAR2	The DN.
lower_case	PLS_INTEGER	If set to 1: The normalized DN returns in lowercase. If set to 0: The case is preserved in the normalized DN string.
norm_dn	VARCHAR2	The normalized DN.

Return Values

The following table lists normalize_dn_with_case function return values

Table 11-44   NORMALIZE_DN_WITH_CASE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

Usage Notes

This function can be used while comparing two DNs.

Related Functions

None

11.9.2 Function get_property_names

The function get_property_names() retrieves the list of property names in the property set.

Syntax

```
FUNCTION get_property_names
(
pset IN PROPERTY_SET,
property_names OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_property_names function parameters:

Table 11-45   GET_PROPERTY_NAMES Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection returned from any of the following functions: • DBMS_LDAP_UTL.get_group_membe rship() • DBMS_LDAP_UTL.get_subscriber_ properties() • DBMS_LDAP_UTL.get_user_proper ties() • DBMS_LDAP_UTL.get_group_prope rties()
property_names	STRING_COLLECTION	A list of property names associated with the property set.

Return Values

The following table lists get_property_names function return values:

Table 11-46   GET_PROPERTY_NAMES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On error.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_property values().

11.9.3 Function get_property_values

The function get_property_values() retrieves the property values (the strings) for a given property name and property.

Syntax

```
FUNCTION get_property_values
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
property_values OUT STRING_COLLECTION
```

```
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_property_values function parameters:

Table 11-47   GET_PROPERTY_VALUES Function Parameters

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	The property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: • DBMS_LDAP_UTL.get_group_membership() • DBMS_LDAP_UTL.get_subscriber_properties() • DBMS_LDAP_UTL.get_user_properties() • DBMS_LDAP_UTL.get_group_properties()
property_values	STRING_COLLECTION	A list of property values (strings).

Return Values

The following table lists get_property_values function return values:

Table 11-48   GET_PROPERTY_VALUES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_property_values_len().

11.9.4 Function get_property_values_len

The function `get_property_values_len()` retrieves the binary property values for a given property name and property.

Syntax

Parameters

Return Values

See Also

Syntax

```
FUNCTION get_property_values_len
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
auth_type IN PLS_INTEGER,
property_values OUT BINVAL_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_property_values_len function parameters:

Table 11-49   GET_PROPERTY_VALUES_LEN Function Parameters

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	A property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: • `DBMS_LDAP_UTL.get_group_membership()` • `DBMS_LDAP_UTL.get_subscriber_properties()` • `DBMS_LDAP_UTL.get_user_properties()` • `DBMS_LDAP_UTL.get_group_properties()`
property_values	BINVAL_COLLECTION	A list of binary property values.

Return Values

The following table lists get_property_values_len function return values:

Table 11-50   GET_PROPERTY_VALUES_LEN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.

Table 11-50   (Cont.) GET_PROPERTY_VALUES_LEN Function Return Values

Value	Description
DBMS_LDAP_UTL.PARAM_ERRO R	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ER ROR	On failure.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_property_values().

11.9.5 Procedure free_propertyset_collection

The procedure free_propertyset_collection() frees the memory associated with property set collection.

Syntax

```
PROCEDURE free_propertyset_collection
(
pset_collection IN OUT PROPERTY_SET_COLLECTION
);
```

Parameters

The following table lists free_propertyset_collection procedure parameters:

Table 11-51   FREE_PROPERTYSET_COLLECTION Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
pset_collection	PROPERTY_SET_COLLECTI ON	The property set collection returned from one of the following functions: - DBMS_LDAP_UTL.get_group_membershi p() - DBMS_LDAP_UTL.get_subscriber_prop erties() - DBMS_LDAP_UTL.get_user_propertie s() - DBMS_LDAP_UTL.get_group_propertie s()

Return Values

None

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(),
DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties().

11.9.6 Function create_mod_propertyset

The function create_mod_propertyset() creates a MOD_PROPERTY_SET data structure.

Syntax

```
FUNCTION create_mod_propertyset
(
pset_type IN PLS_INTEGER,
pset_name IN VARCHAR2,
mod_pset OUT MOD_PROPERTY_SET
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists create_mod_propertyset function parameters:

Table 11-52   CREATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset_type	PLS_INTEGER	The type of property set being modified. Here is a valid value: ENTRY_PROPERTIES
pset_name	VARCHAR2	The name of the property set. This can be NULL if ENTRY_PROPERTIES are being modified.
mod_pset	MOD_PROPERTY_SET	The data structure to contain modify operations to be performed on the property set.

Return Values

The following table lists create_mod_propertyset function return values:

Table 11-53   CREATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.populate_mod_propertyset().

11.9.7 Function populate_mod_propertyset

The function `populate_mod_propertyset()` populates the `MOD_PROPERTY_SET` data structure.

Syntax

```
FUNCTION populate_mod_propertyset
(
mod_pset IN MOD_PROPERTY_SET,
property_mod_op IN PLS_INTEGER,
property_name IN VARCHAR2,
property_values IN STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists populate_mod_propertyset function parameters:

Table 11-54   POPULATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	MOD_PROPERTY_SET	Mod-PropertySet data structure.
property_mod_op	PLS_INTEGER	The type of modify operation to perform on a property. These are valid values: • ADD_PROPERTY • REPLACE_PROPERTY • DELETE_PROPERTY
property_name	VARCHAR2	The name of the property
property_values	STRING_COLLECTION	Values associated with the property.

Return Values

The following table lists populate_mod_propertyset function return values:

Table 11-55   POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.GENERAL_ERROR	Authentication failed.

Table 11-55   (Cont.) POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN	Grace login for user.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.create_mod_propertyset().

11.9.8 Procedure free_mod_propertyset

The procedure free_mod_propertyset() frees the MOD_PROPERTY_SET data structure.

Syntax

```
PROCEDURE free_mod_propertyset
(
mod_pset IN MOD_PROPERTY_SET
);
```

Parameters

The following table lists free_mod_propertyset procedure parameters:

Table 11-56   FREE_MOD_PROPERTYSET Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	PROPERTY_SET	Mod_PropertySet data structure.

Return Values

None

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.create_mod_propertyset().

11.9.9 Procedure free_handle

The procedure `free_handle()` frees the memory associated with the handle.

Syntax

```
PROCEDURE free_handle
(
handle IN OUT HANDLE
);
```

Parameters

The following table lists free_handle procedure parameters:

Table 11-57   FREE_HANDLE Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
handle	HANDLE	A pointer to a handle.

Return Values

None

Usage Notes

None

Related Functions

```
DBMS_LDAP_UTL.create_user_handle(),
DBMS_LDAP_UTL.create_subscriber_handle(),
DBMS_LDAP_UTL.create_group_handle().
```

11.9.10 Function check_interface_version

The function `check_interface_version()` checks the interface version.

Syntax

```
FUNCTION check_interface_version
(
interface_version IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists check_interface_version function parameters:

Table 11-58   CHECK_INTERFACE_VERSION Function Parameters

Parameter Name	Parameter Type	Parameter Description
interface_version	VARCHAR2	Version of the interface.

Return Values

The following table lists check_interface_version return values:

Table 11-59   CHECK_VERSION_INTERFACE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	Interface version is supported.
DBMS_LDAP_UTL.GENERAL_ERROR	Interface version is not supported.

Usage Notes

None

Related Functions

None

11.9.11 Function get_property_values_blob

The function get_property_values_blob() retrieves large binary property values for a given property name and property.

Syntax

```
FUNCTION get_property_values_blob
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
auth_type IN PLS_INTEGER,
property_values OUT BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

The following table lists get_property_values_blob function parameters:

Table 11-60   GET_PROPERTY_VALUES_BLOB Function Parameters

Parameters	Parameter Type	Description
property_name	VARCHAR2	A property name.

Table 11-60　(Cont.) GET_PROPERTY_VALUES_BLOB Function Parameters

Parameters	Parameter Type	Description
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: • DBMS_LDAP_UTL.get_group_membership() • DBMS_LDAP_UTL.get_subscriber_properties() • DBMS_LDAP_UTL.get_user_properties() • DBMS_LDAP_UTL.get_group_properties()
property_values	BLOB_COLLECTION	A list of binary property values.

Return Values

The following table lists get_property_values_blob function return values:

Table 11-61　GET_PROPERTY_VALUES_BLOB Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

Usage Notes

None

Related Functions

DBMS_LDAP_UTL.get_property_values().

11.9.12 Procedure property_value_free_blob

Frees the memory associated with BLOB_COLLECTION returned by
DBMS_LDAP.get_property_values_blob().

Syntax

```
Syntax
PROCEDURE property_value_free_blob
(
vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

The following table lists property_value_free_blob function parameters:

Table 11-62   PROPERTY_VALUE_FREE_BLOB Function Parameters

Parameter	Description
vals	The collection of large binary values returned by DBMS_LDAP.get_property_values_blob().

Return Values

None

Usage Notes

None

Related Functions

DBMS_LDAP.get_property_values_blob().

12

Oracle Directory Integration and Provisioning Java API Reference

As of 10*g* (10.1.4.0.1), Oracle offers Oracle Identity Manager andOracle Directory Integration and Provisioning as complementary provisioning products, optimized for different use cases.

- Oracle Identity Manager, formerly known as Oracle Xellerate IP, is an enterprise provisioning platform designed to manage complex environments with highly heterogeneous technologies that can include directories, databases, mainframes, proprietary technologies, and flat files. Oracle Identity Manager offers full-functioned workflow and policy capabilities along with a rich set of audit and compliance features.

- Oracle Directory Integration and Provisioning, a component of the Identity Management infrastructure, is a meta-directory technology designed to perform directory synchronization and provisioning tasks in a directory-centric environment. Oracle Directory Integration and Provisioning is designed to manage a more homogeneous environment consisting of directories and compatible Oracle products. Oracle Directory Integration and Provisioning performs provisioning tasks by using data synchronization. Oracle Directory Integration and Provisioning offers a small deployment footprint when workflow and a full feature policy engine are not required.

The Oracle Internet Directory SDK includes an Oracle Directory Integration and Provisioning user provisioning API, which enables you to manage users and their application properties in the Oracle Identity Management infrastructure. This chapter describes the main features of the API and explains how to use them.

The following topics provide reference information for the Oracle Directory Integration and Provisioning Java APIs:

- Handling Application Registration and Provisioning Configuration
- Classes to Configure Applications
- Handling User Management Operations
- Setting Up the Debug Mode
- Handling a User Sample Code

12.1 Handling Application Registration and Provisioning Configuration

Applications must register with the provisioning system in order to be recognized as provisionable. They must also create their own configuration in Oracle Internet Directory using the command-line interface. Java classes exist for viewing application configurations.

In order to register with the provisioning system, an application must create a provisioning configuration. After the provisioning configuration exists, the provisioning system identifies the application as directory-enabled and provisionable.

The application must perform the following steps to create a provisioning configuration:

- • [Registering an Application](#)
- • [Understanding Provisioning Configuration](#)

12.1.1 Registering an Application

Oracle applications typically register themselves by using the repository APIs in the `repository.jar` file under `$ORACLE_HOME/jlib`. This file is provided during installation specifically for application registration. In addition to creating an application entry in Oracle Internet Directory, repository APIs can be used to add the application to privileged groups.

Applications written by customers, however, cannot use the `repository.jar` APIs to perform application registration. So application developers must use LDIF templates and create application entries in Oracle Internet Directory using LDAP commands.

An application must create a container for itself under one of these containers:

- • `"cn=Products,cn=OracleContext"`—for applications that service users in multiple realms

- • `"cn=Products,cn=OracleContext,RealmDN"`—for applications that service users in a specific realm

If an application is configured for a specific realm, then that application cannot manage users in other realms. In most cases, you should create the application outside any identity management realm so that the application is not tied to a specific realm in Oracle Internet Directory.

Whenever a new instance of the application installs, a separate entry for the application instance is created under the application's container. Some of the provisioning configuration is common to all the instances of a particular type and some is specific to the instance. When multiple instances of an application are deployed in an enterprise, each instance is independent of the others. Each instance is defined as a separate provisionable application. Users can be provisioned for one or more instances of this application, so that the user can get access to one or more instances of this application.

The examples in this section are for a sample application similar to Oracle Files. When the first instance of this application installs, specific entries must be created in Oracle Internet Directory. In the following example, the name of this application, chosen at run time, is `Files-App1` and the type of the application is `FILES`. The application can have LDIF templates that can be instantiated if required and then uploaded to Oracle Internet Directory. In this example, the application identity is outside any realm. That is, it is under the `"cn=Products,cn=OracleContext"` container.

```
dn: cn=FILES,cn=Products,cn=OracleContext
changetype: add
objectclass: orclContainer

dn: orclApplicationCommonName=Files-App1,cn=FILES,cn=Products,cn=OracleContext
changetype: add
orclappfullname: Files Application Instance 1
userpassword: welcome123
description: This is a test Appliction instance.
protocolInformation: xxxxx
orclVersion: 1.0
orclaci: access to entry by group="cn=odisgroup,cn=DIPAdmins,
 cn=Directory Integration  Platform,cn=Products,
```

```
 cn=OracleContext" (browse,proxy) by group="cn=User Provisioning Admins,
 cn=Groups,cn=OracleContext" (browse,proxy)
orclaci: access to attr=(*) by group="cn=odisgroup,cn=DIPAdmins,
 cn=Directory Integration Platform,cn=Products,
 cn=OracleContext" (search,read,write,compare)
 by group="cn=User Provisioning Admins,
 cn=Groups,cn=OracleContext" (search,read,write,compare)
```

The ACLs shown in the example are discussed in the "Application User Data Location" section.

The application is expected to grant certain privileges to some provisioning services and provisioning administrators.

When the second instance of this application installs, the following entries must be created in Oracle Directory Integration and Provisioning, assuming the name of this application, decided at run time, is `Files-App2`.

```
dn: orclApplicationCommonName=Files-App2,cn=FILES,cn=Products,cn=OracleContext
changetype: add
orclappfullname: Files Application Instance 2
userpassword: welcome123
description: This is a test Appliction instance.
orclVersion: 1.0
orclaci: access to entry by group="cn=odisgroup,
 cn=DIPAdmins,cn=Directory Integration Platform,cn=Products,
 cn=OracleContext" (browse,proxy) by group="cn=User Provisioning Admins,
 cn=Groups,cn=OracleContext" (browse,proxy)
orclaci: access to attr=(*) by group="cn=odisgroup,cn=DIPAdmins,
 cn=Directory Integration Platform,cn=Products,
 cn=OracleContext" (search,read,write,compare) by
 group="cn=User Provisioning Admins,cn=Groups,cn=OracleContext"
 (search,read,write,compare)
```

After the application creates its entries successfully, the application's identity is registered in Oracle Internet Directory. At this point, the application can add itself to certain privileged groups in Oracle Internet Directory, if it needs specific privileges. Table 12-1 shows some of the privileged groups that an application can add itself to. Each of these groups exists in every realm and also in the RootOracleContext. The RootOracleContext Group is a member of the group in all the realms

Table 12-1   Some Useful Privilege Groups

Group Name	Privilege
OracleDASCreateUser	Create a public user
OracleDASEditUser	Edit a public user
OracleDASDeleteUser	Delete a public user
OracleDASCreateGroup	Create a new public group
OracleDASEditGroup	Edit a public group
OracleDASDeleteGroup	Delete a public group

For example, the following LDIF file adds the `Files-App1` application to `cn=OracleCreateUser`, which gives it the privilege to create users in all realms.

```
dn:cn=OracleCreateUser,cn=Groups,cn=OracleContext
changetype: modify
```

```
add: uniquemember
uniquemember: orclApplicationCommonName=Files-
App1,cn=FILES,cn=Products,cn=OracleContext
```

12.1.2 Understanding Provisioning Configuration

An application's provisioning configuration is maintained in its provisioning profile. The provisioning system supports three different provisioning profile versions: Versions 1.1, 2.0 and 3.0. The provisioning service provides different service for the different profile version. Some generic configuration details are common to all applications, regardless of version.

This section contains the following topics:

- Comparison Between Provisioning Configuration Versions
- Version 3.0-Specific Provisioning Configuration Location
- Overview of Provisioning Configuration DIT Structure in Oracle Internet Directory

12.1.2.1 Comparison Between Provisioning Configuration Versions

The differences between the Version 3.0 profile and the Version 2.0 and Version 1.1 profiles are as follows:

- The new provisioning framework recognizes only Version 3.0 applications. Therefore, only applications with provisioning profile Version 3.0 show up as target applications to be provisioned in Oracle Provisioning Console. Applications with Version 2.0 and Version 1.1 profiles do not show them up as applications to be provisioned in the Provisioning Console. Still, the applications are notified about the events that the applications have configured for.

- Creating the provisioning configuration of an application is a multi step process for Version 3.0 profiles. For the earlier version profiles, provisioning registration requires only a single step, running the `oidprovtool` command.

- Applications can subscribe for provisioning events using different interfaces. Two of the interfaces, Java and OID-LDAP, are available only for interface Version 3.0, which is coupled with provisioning configuration Version 3.0. See Table 12-2.

- An application can specify its application-specific user attributes configuration in an LDIF file. This is supported only for interface Version 3.0, which is coupled with provisioning configuration Version 3.0. See Application User Attribute and Defaults Configuration.

- The provisioning status of the user, discussed in the Understanding User Provisioning Status in *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*, is maintained only for Version 3.0 applications. It is not maintained for applications having profiles earlier than Version 3.0.

- Event propagation configuration parameters vary from one version to another. See Table 12-5.

12.1.2.2 Version 3.0-Specific Provisioning Configuration Location

Unless otherwise stated, the remainder of this section describes the Version 3.0-specific provisioning configuration. All the provisioning configuration information is located under the following container:

```
cn=Provisioning,cn=Directory Integration Platform,cn=Products,cn=OracleContext
```

Common provisioning configuration information is stored in entries under the container:

```
cn=Profiles,cn=Provisioning,cn=Directory Integration Platform,
 cn=Products,cn=OracleContext
```

The rest of the provisioning configuration for an application is located under:

```
cn=ApplicationType,cn=Applications,cn=Provisioning,
 cn=Directory Integration Platform,cn=Products,cn=OracleContext
```

All the instances of a specific application type share the configuration under this container. That is, whenever a second instance of an existing application type creates a provisioning profile, all the configuration information under the "cn=*ApplicationType*" container is shared.

12.1.2.3 Overview of Provisioning Configuration DIT Structure in Oracle Internet Directory

Figure 12-1 shows the DIT in Oracle Internet Directory used to store the provisioning configuration.

Figure 12-1   The Directory Information Tree for Provisioning Configuration Data
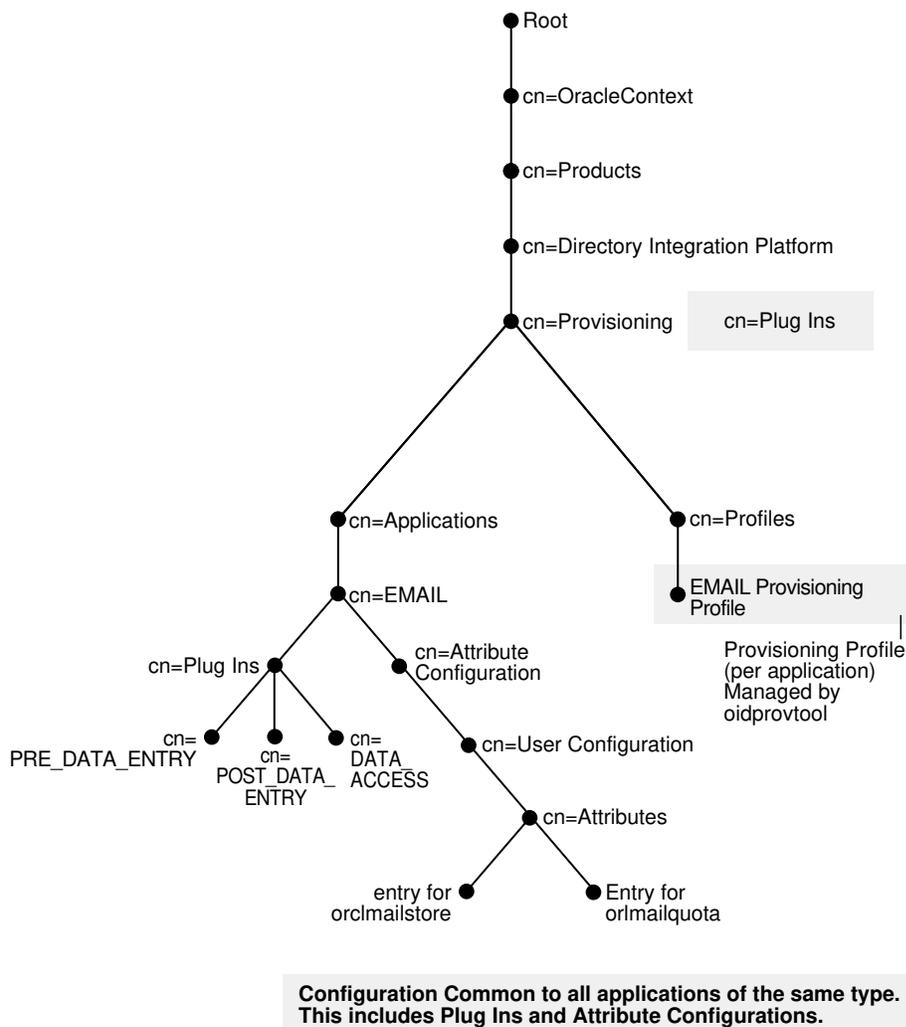
Whenever an instance of an application creates a profile, the new profile is stored as a separate entry under the `Profiles` container in the following naming format:

```
orclODIPProfileName=GUID_of_the_Realm_Entry_GUID_of_the_Application_Identity,….
```

An application must specify the information described in the topics covered in this section when creating a provisioning configuration.

The `Profiles` container contains the following types of configuration information:

- Application Identity Information Parameters
- Application Identity Realm Information
- Default Policy Configuration
- Application User Data Location
- Event Interface Configuration
- Application User Attribute and Defaults Configuration

- Application Provisioning Plug-in Configuration
- Parameters for Event Propagation Configuration
- Application Event Propagation Run Time Status

12.1.2.3.1 Application Identity Information Parameters

An instance of an application is uniquely identified by the following parameters:

- Application DN—A unique DN in the Oracle Internet Directory representing the application. This is a mandatory parameter.
- Application Type— A parameter that is common to all instances of the same application. Multiple instances of a particular type can share some configuration. This is a mandatory parameter.
- Application Name—This can be separately specified. If not specified, it is extracted from the DN. This is an optional parameter.
- Application Display Name—A user-friendly name for the application. This shows up on the Provisioning Console as a target provisionable application. This is an optional parameter.

You provide these application identity parameters while creating the provisioning profile by using the following arguments to the `$ORACLE_HOME/bin/oidprovtool` command line utility, respectively:

- `application_type`
- `application_dn`
- `application_name`
- `application_display_name`

> **See Also:**
>
> The `oidprovtool` command-line tool reference in Standard LDAP Attributes in *Oracle Fusion Middleware Reference for Oracle Identity Management*.

12.1.2.3.2 Application Identity Realm Information

An application registers for a specific realm in order to provide services to the users of that realm only. An application must create a separate provisioning profile for each of the realms it provides services for. In a multi realm scenario, such as a hosted Oracle Portal scenario, applications must register for individual realms.

Whenever a provisioning administrator for a realm accesses the Provisioning Console, only the applications that are registered for that realm are shown as provisionable target applications.

The application specifies realm information while creating the provisioning profile by using the `$ORACLE_HOME/bin/oidprovtool` command line utility with the argument `organization_dn`.

12.1.2.3.3 Default Policy Configuration

While creating a provisioning profile, an application can specify whether the Provisioning Console should manage provisioning to that application or not. If not, the application does not show up on the Provisioning Console as an application to be provisioned. However, Oracle Directory Integration and Provisioning still processes this profile and propagates the events as expected.

An application specifies this information while creating the provisioning profile by using the `application_isdasvisible` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility. The default value is `TRUE`.

An application can configure a default policy determining whether all the users in that realm should be provisioned for that application by default or no users should be provisioned by default. The valid values are

- `PROVISIONING_REQUIRED`—all users are provisioned by default

- `PROVISIONING_NOT_REQUIRED`—no users are provisioned by default

The default is set to `PROVISIONING_REQUIRED`

You can override the default policy with application-provided policy plug-ins at run time. In addition, an administrator can override both the default policy and the decision of the policy plug-in.

An application provides the default policy information by using the `default_provisioning_policy` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility.

12.1.2.3.4 Application User Data Location

Application-specific user information is stored in the application-specific containers. If this data is to be managed by the provisioning system, the application must specify the location of these containers during provisioning registration. An application specifies its user data location by using the `user_data_location` argument to the `$ORACLE_HOME/bin/oidprovtool` command line utility. The application must ensure that the ACLs on this container allow Oracle Delegated Administration Services and Oracle Directory Integration and Provisioning to manage the information in this container.

12.1.2.3.5 Event Interface Configuration

Applications can subscribe for provisioning events using different interfaces: PLSQL, Java, and OID-LDAP. Table 12-2 lists the supported interfaces and their associated configuration. Note that `INTERFACE_VERSION` is coupled with provisioning profile version.

Table 12-2   Interfaces and Their Configuration

Configuration Parameter	PLSQL	Java	OID-LDAP
INTERFACE_VERSION	1.1, 2.0, 3.0	3.0	3.0

Table 12-2   (Cont.) Interfaces and Their Configuration

Configuration Parameter	PLSQL	Java	OID-LDAP
INTERFACE_NAME	The name of the PLSQL package that implements the Interface	Not used	Not used
INTERFACE_CONNECT_INFO	The Database Connect String. Multiple formats supported for all versions.	Not used	Not used
INTERFACE_ADDITIONAL_INFO	Not used	Not used	Not used
Plugin types	PRE_DATA_ENTRY, POST_DATA_ENTRY, DATA_ACCESS	PRE_DATA_ENTRY, POST_DATA_ENTRY, DATA_ACCESS, EVENT_DELIVERY (MUST)	PRE_DATA_ENTRY, POST_DATA_ENTRY, DATA_ACCESS
Description	Mainly for applications that have an Oracle Database back end. The DIP Server pushes the event to the remote Database by invoking the PLSQL procedure.	If the Interface Type is JAVA, an event delivering plug-in must be configured or the server gives errors. The plug-in configuration determines the rest of the configuration. See Application Provisioning Plug-in Configuration.	Mainly used in cases where the application is very tightly bound to Oracle Internet Directory and event delivery through the PLSQL interface or the JAVA Event Delivery Plug-in is unnecessary. This interface is deprecated in future. Please use the JAVA Interface instead.

Applications can use the following arguments to `$ORACLE_HOME/bin/oidprovtool` when specifying an event interface configuration:

- `interface_type` (Default is PLSQL)

- `interface_version` (Default is 2.0)

- `interface_name`

- `interface_connect_info`

- `interface_additional_info`

Table 12-3 lists the interface connection information formats that the PL/SQL interface supports when it connects to a remote database. All the formats are supported for all interface versions.

Table 12-3   Information Formats Supported by the PLSQL Interface

Format	Description
scanListener_dbHost:*scanListener_dbPort*:*scanListener_Long_dbServiceName*:*username*:*password*	Recommended for High Availability implementations where RAC is configured using a Scan Listener. DIP passes this to the thin JDBC Driver.
DBURL=ldap://*LDAP_host*:*LDAP_port*/ *ServiceName*,cn=OracleContext,..:*username*:*password*	Required for High Availability environments where RAC is configured using Virtual IP (VIP). DIP passes this to the thin JDBC Driver and the driver looks up the Database Registration entry in Oracle Internet Directory to get the actual Database connection information.

Some examples of supported formats are:

```
scanlisthost.example.com:1521:iasdbsvc:scott:tiger
```

```
DBURL=ldap://example.com:3060/
samplegdbname,cn=OracleContext,dc=example,dc=com:scott:tiger
```

12.1.2.3.6 Application User Attribute and Defaults Configuration

An application can specify its application-specific user attributes configuration in an LDIF file. This is supported only for interface version 3.0.

As shown in Figure 12-1, the configuration for a particular attribute is stored as a separate entry under the container:

```
"cn=Attributes,cn=User Configuration,cn=Attribute configuration,
 cn=Application_Type,cn=Applications,cn=Provisioning,
 cn=Directory Integration Platform,cn=Products,cn=OracleContext"
```

There is no argument to `oidprovtool` for uploading this information. The application must use an LDAP file and command-line tools to upload its attribute configuration information to Oracle Internet Directory.

Each application-specific attribute is represented as a separate entry. The following example is for the attribute `orclFilesDomain`:

```
dn: cn=orclFilesDomain,cn=Attributes,cn=User configuration,cn=Attribute
configuration,……
changetype: add
orcldasadminmodifiable: 1
orcldasviewable: 1
displayname: Files Domain
orcldasismandatory: 1
orcldasuitype: LOV
orcldaslov: us.example.com
orcldaslov: uk.example.com
orclDASAttrIsUIField: 1
orclDASAttrIsFieldForCreate: 1
orclDASAttrIsFieldForEdit: 1
orclDASAttrToDisplayByDefault: 1
orclDASSelfModifiable: 1
orclDASAttrDisplayOrder: 1
```

```
orclDASAttrDefaultValue: us.example.com
orclDASAttrObjectClass: orclFILESUser
objectclass: orclDASConfigAttr
objectclass: orclcontainer
```

Table 12-4 explains the significance of each of the properties that are stored as attributes in the attribute configuration entry.

Table 12-4   Properties Stored as Attributes in the Attribute Configuration Entry

Property Name	Description	Comments
orclDASIsUIField	Whether this property is to be shown in the DAS Console or not	Not Used in 11g Release 1 (11.1.1). All attributes are shown.
orclDASUIType	The Type of the UI Field: singletext, multitext, LOV, DATE, Number, password	Used byOracle Internet Directory Self-Service Console only
orclDASAdminModifiable	Whether the field is modifiable by the administrator or not	Not Used in 11g Release 1 (11.1.1). All attributes are modifiable by administrator.
orclDASViewAble	Whether this attribute is a read-only attribute in the Oracle Internet Directory Self-Service Console	Not Used in 11g Release 1 (11.1.1)
displayName	The Localized Name of the attribute as it shows on the Oracle Internet Directory Self-Service Console	
orclDASIsMandatory	Whether this attribute is mandatory or not	If a mandatory attribute is not populated, the Oracle Internet Directory Self-Service Console complains
orclDASAttrIsFieldForCreate	Whether to expose this attribute only during user creation	Not Used in 11g Release 1 (11.1.1)
orclDASAttrIsFieldForEdit	Whether to expose this attribute only during user editing	Not Used in 11g Release 1 (11.1.1)
orclDASAttrToDisplayByDefault	Whether to hide the attribute by default under a collapsed section	Not Used in 11g Release 1 (11.1.1)
orclDASSelfModifiable	Whether this attribute is modifiable by the user or not	Not Used in 11g Release 1 (11.1.1), as Oracle Internet Directory Self-Service Console is only for application-specific attributes. Users cannot change their user preferences from the Oracle Internet Directory Self-Service Console.
OrclDASAttrDisplayOrder	The order is which the attribute is to be displayed in the application-specific section	Not Used in 11g Release 1 (11.1.1)

Table 12-4   (Cont.) Properties Stored as Attributes in the Attribute Configuration Entry

Property Name	Description	Comments
OrclDASAttrDefaultValue	The initial default value for the attribute that is used by the provisioning components: Oracle Internet Directory Self-Service Console, Oracle Directory Integration and Provisioning, Bulk Provisioning Tool	Can be changed using the Oracle Internet Directory Self-Service Console *Application Management Page*. The Plug-ins or the administrator can override the initial default values.
OrclDASAttrObjectClass	The LDAP object class that the attribute belongs to.	Used to create the application-specific user entries that the provisioning system maintains.

If an application has application-specific attributes, you can specify that the provisioning system manage its attributes defaults. You do that by using the `manage_application_defaults` argument to `$ORACLE_HOME/bin/oidprovtool`. This argument is `TRUE` by default.

12.1.2.3.7 Application Provisioning Plug-in Configuration

Application provisioning plug-ins are discussed in

Java Plug-ins for User Provisioning .

12.1.2.3.8 Parameters for Event Propagation Configuration

Event propagation configuration parameters vary from one profile version to another. Table 12-5 lists and describes configuration parameters for event propagation.

Table 12-5   Event propagation parameters

Parameter	Supported Provisioning Profile Version	Description
profile_mode	2.0,.3.0	Whether the application is to receive outbound provisioning events from Oracle Internet Directory, to send inbound events, or both. Values are OUTBOUND (default), INBOUND, and BOTH.
Schedule	1.1, 2.0, 3.0	The scheduling interval after which pending events are propagated
enable_bootstrap	3.0	Enables events for application bootstrapping. This specifies that the application should be notified of users that existed in Oracle Internet Directory before the application created its provisioning profile.

Table 12-5   (Cont.) Event propagation parameters

Parameter	Supported Provisioning Profile Version	Description
enable_upgrade	3.0	Enables events for application user upgrade. This specifies that the application should be notified of users that existed in Oracle Internet Directory before the upgrade. If the application was present before the upgrade, users might already exist in the application. For such users, Oracle Directory Integration and Provisioning sends an Upgrade Event to the application so that the user is handled differently from a normal new user.
lastchangenumber	3.0	The change number in Oracle Internet Directory from which the events need to be sent to the application.
max_prov_failure_li mit	3.0	The maximum number of retries that the Oracle Directory Integration and Provisioning server attempts when provisioning a user for that application.
max_events_per_in vocation	2.0, 3.0	For bulk event propagation, this specifies the maximum number of events that can be packaged and sent during one invocation of the event interface.
max_events_per_s chedule	2.0	Maximum number of events that Oracle Directory Integration and Provisioning sends to an application in one execution of the profile. The default is 25. In deployments with many profiles and applications, this enables Oracle Directory Integration and Provisioning, which is multithreaded, to execute threads for multiple profiles.

event_subscription	1.1, 2.0, 3.0	Defines the types of OUTBOUND events an application is to receive from the event propagation service. The format is:
		
		Object_Type:*Domain*:*Operation*(*Attributes*,…)
		
		For example:
		
		`USER:cn=users,dc=example,dc=com:ADD(*)`
		
		specifies that USER_ADD event should be sent if the user that was created is under the specified domain and that all attributes should also be sent.
		
		`USER:cn=users,dc=example,dc=com:MODIFY(cn,sn.m ail,telephonenumber)`
		
		specifies that USER_MODIFY event should be sent if the user that was modified is under the specified domain and any of the listed attributes were modified
		
		`USER:cn=users,dc=example,dc=com:DELETE`
		
		specifies that USER_DELETE event should be sent if a user under the specified domain was deleted

Table 12-5   (Cont.) Event propagation parameters

Parameter	Supported Provisioning Profile Version	Description
event_permitted_op erations	2.0	Defines the types of INBOUND events an application is privileged to send to the Oracle Directory Integration and Provisioning server. The format is:
		`Object_Type:Domain:Operation(Attributes,…)`
		For example:
		`IDENTITY:cn=users,dc=example,dc=com:ADD(*)`
		specifies that IDENTITY_ADD event is allowed for the specified domain and all attributes are also allowed. This means that the application is allowed to create users in Oracle Internet Directory.
		`IDENTITY:cn=users,dc=example,dc=com:MODIFY(cn, sn.mail,telephonenumber)`
		Specifies that IDENTITY_MODIFY is allowed for only the attributes in the list. Other attributes are silently ignored. This means that the application is allowed to modify the listed attributes of the users in Oracle Internet Directory.
		`IDENTITY:cn=users,dc=example,dc=com:DELETE`
		Specifies that the application is allowed to delete users in Oracle Internet Directory
event_mapping_rul es	2.0	For INBOUND profiles, this specifies the type of object received from an application and a qualifying filter condition to determine the domain of interest for this event. Multiple rules are allowed. The format is:
		`Object_Type: Filter_condition: Domain_Of_Interest`
		For example:
		`EMP::cn=users,dc=example,dc=com`
		specifies that if the object type received is `EMP`, the event is meant for the domain "`cn=users,dc=example,dc=com`".
		`EMP:l=AMERICA:l=AMER,cn=users,dc=example,dc=com`
		specifies that if the object type received is `EMP`, and the event has the attribute `l` (locality) and its value is `AMERICA`, the event is meant for the domain "`l=AMER,cn=users,dc=example,dc=com`".

12.1.2.3.9 Application Event Propagation Run Time Status

The Oracle Provisioning Service records a user's provisioning status in Oracle Internet Directory for each provisioning-integrated application. This is described in the Deploying and Configuring Provisioning chapter of *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*.

12.2 Classes to Configure Applications

The `oracle.idm.user.provisioning.configuration.Configuration` class enables you to obtain provisioning schema information.

The `oracle.idm.user.provisioning.configuration.Application` class enables you to obtain metadata for registered applications. These classes are documented under the package `oracle.idm.provisioning.configuration`.

The `Configuration` class provides access to application configurations. To construct, a `Configuration` object, you must specify the realm. For example:

```
Configuration cfg = new Configuration ("us");
```

Then you use `Configuration` class methods to get one or all application configurations in a realm. You must supply the LDAP context of the realm.

The `Configuration` object is a fairly heavy weight object, as its creation requires access to the Oracle Internet Directory metadata. Best practice is to create a `Configuration` object once during initialization of an application, then to reuse it for all operations that require it.

The `Application` object represents an application instance. Its methods provide metadata about a registered application in the infrastructure.

12.3 Handling User Management Operations

When Oracle Directory Integration and Provisioning or Oracle Delegated Administration Services invokes a provisioning plug-in, it passes information about the user being provisioned. A deployed application can use the user object to modify the user.

The user management provisioning classes provide the following operations:

* Create, modify, and delete a base user
* Create, modify, and delete application-specific user information
* Search base users
* Retrieve user provisioning status for applications

This section includes the following topics:

* Creating a User
* Modifying a User
* Deleting a User
* Looking Up a User

12.3.1 Creating a User

The combination of the base user and application user in the repository is referred to as the Oracle Identity Management user. The minimum information required to create a user is a set of attributes representing the base user. The attributes are in the form of name-value pairs. These user attributes are represented as Java objects using the class `oracle.ldap.util.ModPropertySet.`

Creating a user in theOracle Identity Management repository consists of two steps:

1. Creating basic user information in the specified realm. This information is referred to as the base user.

2. Creating the application-specific user attributes, or footprint. This information is referred to as the application user.

Some methods create only the base user and other create both components of the Oracle Identity Management user.

Some user creation methods require you to specify the DN of the entry that you want to create in the Oracle Identity Management user repository. Other methods do not require the DN. Instead, they construct the Oracle Identity Management user using the metadata configuration information from the Realm in which the user is created.

If the creation of the base user and application user succeeds, then the creation method returns an `IdmUser` object. You use this object to manage the attributes of the base user and application user.

12.3.2 Modifying a User

You can modify a base user in Oracle Identity Management repository.

Modifying a base user in the Oracle Identity Management repository results in

- Modifying the base user information

- Creating or modifying application user information

You must supply the following information in order to modify an Oracle Identity Management user:

1. The user's DN, GUID, or `IdmUser` object reference

2. The desired changes to the base user attributes, represented as an `oracle.ldap.util.ModPropertySet`

Some user modification methods modify only the base user attributes. Others modify the application user attributes as well.

12.3.3 Deleting a User

You can delete a base user in Oracle Identity Management repository.

Deleting a base user in theOracle Identity Management repository produces the following results:

- Deleting the base user information

- Deleting the application user information

To modify anOracle Identity Management user, you must supply the DN, GUID, or IdmUser object reference.

As result of this operation, the base user and the application user attributes are deleted.

12.3.4 Looking Up a User

In order to look up Oracle Identity Management users, you must provide the DN or GUID.

The lookup methods provide two lookup options:

- Look up a specific Oracle Identity Management user using GUID or DN
- Look up a set of Oracle Identity Management users using a search filter

The output of a lookup method is one of the following:

- A single IdmUser object
- A list of IdmUser objects

12.4 Setting Up the Debug Mode

Set `UtilDebug.MODE_PROVISIONING_API` mode to enable debugging and trace information. If you do not specify an output stream for the log messages, they are written to standard output.

The following snippet shows how to set `UtilDebug.MODE_PROVISIONING_API` mode and specify an output stream:

```
Import oracle.ldap.util.UtilDebug;
    FileOutputStream logStream = new FileOutputStream("ProvAPI.log")
    …
    UtilDebug.setDebugMode(UtilDebug.MODE_PROVISIONING_API);
UtilDebug.setPrintStream(logStream);
```

12.5 Handling a User Sample Code

You can create, modify and look up a user and get user provisioning status.

The following code example shows how to create, modify, and look up a user and how to get user provisioning status for an application:

```
UtilDebug.setDebugMode(UtilDebug.MODE_PROVISIONING_API);
…
Configuration cfg = new Configuration(realm);
    try {
      debug("Connecting...");
      InitialLdapContext ctx =
          ConnectionUtil.getDefaultDirCtx(hostName, port, bindDn, passwd);
      debug("Connected...");
      UserFactory factory = UserFactoryBuilder.createUserFactory(ctx, cfg);

      // Create
      ModPropertySet mpSet = new ModPropertySet();
      mpSet.addProperty("cn","Heman");
      mpSet.addProperty("sn","The Master");
      mpSet.addProperty("uid","Heman");
```

```
        IdmUser idmUser = factory.createUser(mpSet);

        // Modify
        mpSet = new ModPropertySet();
        mpSet.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_REPLACE,"sn",
              "Heman The Master");
        mpSet.addProperty("givenName","Master of the Universe");
        factory.modifyUser(idmUser, mpSet);


            // Lookup          List users =
    factory.searchUsers(Util.IDTYPE_SIMPLE, "Hema*", null);
        ….

            // Get user provisioning status for an application.
            Application app = cfg.getApplication(lCtx, "Files", "FilesInstace");
            String  status = idmUser.getProvisioningStatus(app);


        // Another way to get user provisioning status
            String userDn = idmUser.getDNn();
            String  status = ProvUtil.getUserProvisioningStatus(dirctx,
              Util.IDTYPE_DN, userDn, app.getType(), app.getName());
      } catch (Exception ex) {
        ex.printStackTrace();
         //
      }
```

13

Oracle Directory Integration Platform PL/SQL API Reference

You need to know about interface versions and Extensible Event Definition Configuration to work on Oracle Directory Integration Platform PL/SQL API reference.
The following topics describe the registration API for the Oracle Directory Provisioning Integration Service:

- About Versions for Provisioning Files and Interfaces
- Understanding Extensible Event Definition Configuration
- Attributes of Inbound and Outbound Events
- Implementing PL/SQL Bidirectional Interface (Version 3.0)
- Implementing PL/SQL Bidirectional Interface (Version 2.0)
- Provisioning Event Interface (Version 1.1)

13.1 About Versions for Provisioning Files and Interfaces

Various versions of provisioning files and interfaces is discussed in the following section.

In release 9.0.2, the default interface version was version 1.1. In releases 9.0.4 and 10.1.2.0.0, the interface version defaults to version 2.0. Release 10.1.2.0.1 adds yet a third version. The administrator can use any one of these.

13.2 Understanding Extensible Event Definition Configuration

This feature is only for outbound events. It addresses the ability to define a new event at run time so that the provisioning integration service can interpret a change in Oracle Internet Directory and determine whether an appropriate event is to be generated and propagated to an application.

The following events are the only configured events at installation time.The following topics describe the Event Definition feature:

- Attributes of an Event Definition
- LDAP Configuration of a Predefined Event
- Defining a New Event

13.2.1 Attributes of an Event Definition

Event object type consists of Event object type, LDAP change type and Event criteria attributes.

An event definition (entry) attributes are described as below:

- Event object type (`orclODIPProvEventObjectType`): This specifies the type of object the event is associated with. For example, the object could be a `USER`, `GROUP`, or `IDENTITY`.

- LDAP change type (`orclODIPProvEventChangeType`): This indicates that all kinds of LDAP operations can generate an event for this type of object. (e.g `ADD`, `MODIFY`, `DELETE`)

- Event criteria (`orclODIPProvEventCriteria`): The additional selection criteria that qualify an LDAP entry to be of a specific object type. For example, `Objectclass=orclUserV2` means that any LDAP entry that satisfies this criteria can be qualified as this Object Type and any change to this entry can generate appropriate events.

The object class that holds these attributes is `orclODIPProvEventTypeConfig`. The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event type configurations.

Table 13-1 lists the event definitions predefined as a part of the installation.

Table 13-1   Predefined Event Definitions

Event Object Type	LDAP Change Type	Event Criteria
ENTRY	ADD MODIFY DELETE	objectclass=*
USER	ADD MODIFY DELETE	objectclass=interorgperson objectclass=orcluserv2
IDENTITY	ADD MODIFY DELETE	objectclass=interorgperson objectclass=orcluserv2
GROUP	ADD MODIFY DELETE	objectclass=orclgroup objectclass=groupofuniquenames
SUBSCRPTION	ADD MODIFY DELETE	objectclass=orclservicerecepient
SUBSCRIBER	ADD MODIFY DELETE	objectclass=orclsubscriber

13.2.2 LDAP Configuration of a Predefined Event

The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event definition configurations.

LDAP configuration of the predefined event definitions is as follows:

```
dn: orclODIPProvEventObjectType=ENTRY,cn=ProvisioningEventTypeConfig,cn=odi, cn=oracle
internet directory
orclODIPProvEventObjectType: ENTRY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=*
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=USER,cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle
internet directory
orclODIPProvEventObjectType: USER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=InetOrgPerson
orclODIPProvEventCriteria: objectclass=orcluserv2
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=IDENTITY,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: IDENTITY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=inetorgperson
orclODIPProvEventCriteria: objectclass=orcluserv2
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=GROUP,cn=ProvisioningEventTypeConfig,cn=odi, cn=oracle
internet directory
orclODIPProvEventObjectType: GROUP
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclgroup
orclODIPProvEventCriteria: objectclass=groupofuniquenames
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=SUBSCRIPTION,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIPTION
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclservicerecepient
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=SUBSCRIBER,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIBER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclsubscriber
objectclass: orclODIPProvEventTypeConfig
```

13.2.3 Defining a New Event

To define a new event of Object type XYZ (which is qualified with the object class `objXYZ`), create the following entry in Oracle Internet Directory. The DIP server recognizes this new event definition and propagates events if necessary to applications that subscribe to this event.

```
dn: orclODIPProvEventObjectType=XYZ,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: XYZ
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=objXYZ
objectclass: orclODIPProvEventTypeConfig
```

This means that if an LDAP entry with the object class `objXYZ` is added, modified, or deleted, DIP propagates the `XYZ_ADD`, `XYZ_MODIFY`, or `XYZ_DELETE` event to any application concerned.

13.3 Attributes of Inbound and Outbound Events

An application can register as a supplier as and as a consumer of events.

The provisioning subscription profile has the attributes described in Table 13-2.

Table 13-2   Attributes of the Provisioning Subscription Profile

Attribute	Description
EventSubscriptions	Outbound events only (multivalued).
	Events for which DIP should send notification to this application. The format of this string is `[USER]GROUP]:[`*`domain_of_interest`*`]:[DELETE|ADD|MODIFY(`*`list_of_attributes_separated_by_comma`*`)]`
	Multiple values may be specified by listing the string multiple times, each time with different values. If parameters are not specified, the following defaults are assumed: `USER:`*`organization_DN`*`:DELETEGROUP:`*`organization_DN`*`:DELETE`—that is, send user and group delete notifications under the organization DN.
MappingRules	Inbound events Only (multivalued).
	This attribute is used to map the type of object received from an application and a qualifying filter condition to determine the domain of interest for this event. The mapping takes this form:
	`OBJECT_TYPE``: `*`Filter_condition`*`: `*`domain_of_interest`*
	Multiple rules are allowed. In the mapping `EMP:cn=users,dc=example,dc=com`, the object type received is `EMP`. The event is meant for the domain `cn=users,dc=example,dc=com`. In the mapping `EMP:l=AMERICA:l=AMER,cn=users,dc=example,dc=com`, the object type received is `EMP`. The event is meant for the domain `l=AMER,cn=users,dc=example,dc=com`.

Table 13-2   (Cont.) Attributes of the Provisioning Subscription Profile

Attribute	Description
`permittedOperations`	Inbound events only (multi valued).
	This attribute is used to define the types of events an application is privileged to send to the provisioning integration service. The mapping takes this form:
	Event_Object: *affected_domain*:*operation*(*attributes*, . . .)
	In the mapping `IDENTITY:cn=users,dc=example,dc=com:ADD(*)` the `IDENTITY_ADD` event is allowed for the specified domain and all attributes are also allowed. In the mapping `IDENTITY:cn=users,dc=example,dc=com:MODIFY(cn,sn.mail,telephonenumber)`, the `IDENTITY_MODIFY` event is allowed only for the attributes in the list. Any extra attributes are silently ignored.

13.4 Implementing PL/SQL Bidirectional Interface (Version 3.0)

The PL/SQL callback interface requires you to develop a PL/SQL package that Oracle Directory Provisioning Integration Service invokes in the application specific database.

Before attempting to use Version 3.0 of the PL/SQL interface, please refer to:

- Java Plug-ins for User Provisioning

- The chapter Understanding Oracle Directory Integration and Provisioning for Provisioning in*Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*

- The chapter Deploying Provisioning-Integrated Applications in *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*.

Choose any name for PL/SQL package, but be sure to use the same name when you register the package at subscription time. Implement the package by using the following PL/SQL package specification:

```
DROP TYPE LDAP_EVENT_LIST_V3;
DROP TYPE LDAP_EVENT_V3;
DROP TYPE LDAP_EVENT_STATUS_LIST_V3;
DROP TYPE LDAP_ATTR_LIST_V3;
DROP TYPE LDAP_ATTR_V3;
DROP TYPE LDAP_ATTR_VALUE_LIST_V3;
DROP TYPE LDAP_ATTR_VALUE_V3;


-----------------------------------------------------------------------------------
-----------
-- Name: LDAP_ATTR_VALUE_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains values of an attribute. A list of one or
more of this object is passed in any event.
-----------------------------------------------------------------------------------
------------

CREATE TYPE LDAP_ATTR_VALUES_V3 AS OBJECT (
    attr_value       VARCHAR2(4000),
    attr_bvalue      RAW(2048),
    attr_value_len   INTEGER
);
```

```
GRANT EXECUTE ON LDAP_ATTR_VALUE_V3 to public;

CREATE TYPE LDAP_ATTR_VALUE_LIST_V3 AS TABLE OF LDAP_ATTR_VALUE_V3;
/
GRANT EXECUTE ON LDAP_ATTR_VALUE_LIST_V3 to public;


--------------------------------------------------------------------------------
-----------------
-- Name: LDAP_ATTR_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains details regarding an attribute. A list of
one or more of this object is passed in any event.
--------------------------------------------------------------------------------
-----------------
CREATE TYPE LDAP_ATTR_V3 AS OBJECT (
     attr_name        VARCHAR2(256),
     attr_type        INTEGER ,
     attr_mod_op      INTEGER,
     attr_values      LDAP_ATTR_VALUE_LIST_V3
);

GRANT EXECUTE ON LDAP_ATTR_V3 to public;

CREATE TYPE LDAP_ATTR_LIST_V3 AS TABLE OF LDAP_ATTR_V3;
/
GRANT EXECUTE ON LDAP_ATTR_LIST_V3 to public;


--------------------------------------------------------------------------------
-----------------
-- Name: LDAP_EVENT_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains event information plus the attribute
List.
--------------------------------------------------------------------------------
-----------------

CREATE TYPE LDAP_EVENT_V3 AS OBJECT (
          event_type  VARCHAR2(32),
          event_id    VARCHAR2(32),
          event_src   VARCHAR2(1024),
          event_time  VARCHAR2(32),
          object_name VARCHAR2(1024),
          object_type VARCHAR2(32),
          object_guid VARCHAR2(32),
          object_dn   VARCHAR2(1024),
          profile_id  VARCHAR2(1024),
          attr_list   LDAP_ATTR_LIST_V3 ) ;
/

GRANT EXECUTE ON LDAP_EVENT_V3 to public;
CREATE TYPE LDAP_EVENT_LIST_V3 AS TABLE OF LDAP_EVENT_V3;
/
GRANT EXECUTE ON LDAP_EVENT_LIST_V3 to public;


--------------------------------------------------------------------------------
-----------------
-- Name: LDAP_EVENT_STATUS_V3
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains information that is sent by the consumer
of an event to the supplier in response to the actual event.
```

```
-----------------------------------------------------------------------------------------
------------

CREATE TYPE LDAP_EVENT_STATUS_V3 AS OBJECT (
        event_id     VARCHAR2(32),
        status       VARCHAR2(32),
        status_msg   VARCHAR2(2048),
        object_guid  VARCHAR(32)
) ;
/

GRANT EXECUTE ON LDAP_EVENT_STATUS_V3 to public;
CREATE TYPE LDAP_EVENT_STATUS_LIST_V3 AS TABLE OF LDAP_EVENT_STATUS_V3;
/
GRANT EXECUTE ON LDAP_EVENT_STATUS_LIST_V3 to public;


-----------------------------------------------------------------------------------------
------------
-- Name: LDAP_NTFY
-- DESCRIPTION: This is the interface to be implemented by provisioning integrated
applications to send information to and receive information from the directory.
The name of the package can be customized as needed. The function and procedure
names within this package should not be changed.

-----------------------------------------------------------------------------------------
------------

CREATE OR REPLACE PACKAGE LDAP_NTFY AS

    -- The Predefined Event Types

    ENTRY_ADD      CONSTANT VARCHAR2 (32) :='ENTRY_ADD';
    ENTRY_DELETE   CONSTANT VARCHAR2 (32) :='ENTRY_DELETE';
    ENTRY_MODIFY   CONSTANT VARCHAR2 (32) :='ENTRY_MODIFY';

    USER_ADD       CONSTANT VARCHAR2 (32) :='USER_ADD';
    USER_DELETE    CONSTANT VARCHAR2 (32) :='USER_DELETE';
    USER_MODIFY    CONSTANT VARCHAR2 (32) :='USER_MODIFY';

    IDENTITY_ADD      CONSTANT VARCHAR2 (32) :='IDENTITY_ADD';
    IDENTITY_DELETE   CONSTANT VARCHAR2 (32) :='IDENTITY_DELETE';
    IDENTITY_MODIFY   CONSTANT VARCHAR2 (32) :='IDENTITY_MODIFY';

    GROUP_ADD      CONSTANT VARCHAR2 (32) :='GROUP_ADD';
    GROUP_DELETE   CONSTANT VARCHAR2 (32) :='GROUP_DELETE';
    GROUP_MODIFY   CONSTANT VARCHAR2 (32) :='GROUP_MODIFY';

    SUBSCRIPTION_ADD     CONSTANT VARCHAR2(32) :='SUBSCRIPTION_ADD';
    SUBSCRIPTION_DELETE  CONSTANT VARCHAR2(32) :='SUBSCRIPTION_DELETE';
    SUBSCRIPTION_MODI    CONSTANT VARCHAR2(32) :='SUBSCRIPTION_MODIFY';

    SUBSCRIBER_ADD     CONSTANT VARCHAR2(32) :='SUBSCRIBER_ADD';
    SUBSCRIBER_DELETE  CONSTANT VARCHAR2(32) :='SUBSCRIBER_DELETE';
    SUBSCRIBER_MODIFY  CONSTANT VARCHAR2(32) :='SUBSCRIBER_MODIFY';

    -- The Attribute Type

    ATTR_TYPE_STRING            CONSTANT NUMBER  := 0;
    ATTR_TYPE_BINARY            CONSTANT NUMBER  := 1;
    ATTR_TYPE_ENCRYPTED_STRING  CONSTANT NUMBER  := 2;
```

```
     -- The Attribute Modification Type

     MOD_ADD      CONSTANT NUMBER  := 0;
     MOD_DELETE   CONSTANT NUMBER  := 1;
     MOD_REPLACE  CONSTANT NUMBER  := 2;


     -- The Event dispostions constants

     EVENT_SUCCESS            CONSTANT VARCHAR2(32)  :='EVENT_SUCCESS';
     EVENT_IN_PROGRESS        CONSTANT VARCHAR2(32)  :='EVENT_IN_PROGRESS';
     EVENT_USER_NOT_REQUIRED  CONSTANT VARCHAR2(32)  :='EVENT_USER_NOT_REQUIRED';
     EVENT_ERROR              CONSTANT VARCHAR2(32)  :='EVENT_ERROR';
     EVENT_ERROR_ALERT        CONSTANT VARCHAR2(32)  :='EVENT_ERROR_ALERT';
     EVENT_ERROR_ABORT        CONSTANT VARCHAR2(32)  :='EVENT_ERROR_ABORT';


     -- The Actual Callbacks

     FUNCTION GetAppEvents (events OUT LDAP_EVENT_LIST_V3)
     RETURN NUMBER;

     -- Return CONSTANTS
     EVENT_FOUND      CONSTANT NUMBER:= 0;
     EVENT_NOT_FOUND  CONSTANT NUMBER:= 1403;
```

If the provisioning server is unable to process an inbound event, it triggers an EVENT_ERROR_ALERT status, which generates a trigger in Oracle Enterprise Manager.

If the provisioning server is able to process the event, but finds that the event cannot be processed—for example, the user to be modified, subscribed, or deleted does not exist—it responds with EVENT_ERROR to indicate to the application that something is wrong. It is again up to the application to handle the status event.

EVENT_ERROR means no errors in directory operations. The event cannot be processed for other reasons.

```
-- PutAppEventStatus() : DIP Server invokes this callback in the remote Data
base after processing an event it had received using the GetAppEvents()
callback.  For every event received, the DIP server sends the status event
back after processing the event.  This API will NOT be required by the
Oracle Collaboration Suite release 3.0 components.

PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS_LIST_V3);

-- PutOIDEvents() : DIP Server invokes this API in the remote Database. DIP
server sends event to applications using this callback. It also expects a status
event object in response as an OUT parameter. This API needs to be implemented
by all the Oracle Collaboration Suite release 3.0 components.

PROCEDURE PutOIDEvents (event         IN  LDAP_EVENT_LIST_V3,
                        event_status  OUT LDAP_EVENT_STATUS_LIST_V3);

END LDAP_NTFY;
/
```

13.5 Implementing PL/SQL Bidirectional Interface (Version 2.0)

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database.

Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```
DROP TYPE LDAP_EVENT;
DROP TYPE LDAP_EVENT_STATUS;
DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;
-----------------------------------------------------------------------------
-- Name: LDAP_ATTR
-- Data Type: OBJECT

DESCRIPTION: This structure contains details regarding an attribute. A list of one
--            or more of this object is passed in any event.
------------------------------------------------------------------------------------
------------
CREATE TYPE LDAP_ATTR AS OBJECT (
     attr_name        VARCHAR2(256),
     attr_value       VARCHAR2(4000),
     attr_bvalue      RAW(2048),
     attr_value_len   INTEGER,
     attr_type        INTEGER ,
     attr_mod_op      INTEGER
);

GRANT EXECUTE ON LDAP_ATTR to public;

CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
GRANT EXECUTE ON LDAP_ATTR_LIST to public;


------------------------------------------------------------------------------------
------------
-- Name: LDAP_EVENT
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains event information plus the attribute
--              list.
------------------------------------------------------------------------------------
------------

CREATE TYPE LDAP_EVENT AS OBJECT (
         event_type  VARCHAR2(32),
         event_id    VARCHAR2(32),
         event_src   VARCHAR2(1024),
         event_time  VARCHAR2(32),
         object_name VARCHAR2(1024),
         object_type VARCHAR2(32),
         object_guid VARCHAR2(32),
         object_dn   VARCHAR2(1024),
         profile_id  VARCHAR2(1024),
         attr_list   LDAP_ATTR_LIST ) ;
/

GRANT EXECUTE ON LDAP_EVENT to public;
```

```
----------------------------------------------------------------------------------
------------------
-- Name: LDAP_EVENT_STATUS
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains information that is sent by the
--             consumer of an event to the supplier in response to the
--             actual event.

----------------------------------------------------------------------------------
-----------------

CREATE TYPE LDAP_EVENT_STATUS AS OBJECT (
        event_id          VARCHAR2(32),
        orclguid          VARCHAR(32),
        error_code        INTEGER,
        error_String      VARCHAR2(1024),
        error_disposition VARCHAR2(32)) ;
/

GRANT EXECUTE ON LDAP_EVENT_STATUS to public;
```

13.6 Provisioning Event Interface (Version 1.1)

You must develop logic to consume events generated by the provisioning integration service. The interface between the application and the provisioning integration service can be table-based, or it can use PL/SQL callbacks.

The following sections describes PL/SQL callbacks:

- Using PL/SQL Callback Interface
- Predefined Event Types
- Attribute Type
- Attribute Modification Type
- Event Dispositions Constants
- About Callbacks

13.6.1 Using PL/SQL Callback Interface

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database.

Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```
Rem
Rem      NAME
Rem         ldap_ntfy.pks - Provisioning Notification Package Specification.
Rem

DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;

-- LDAP ATTR
----------------------------------------------------------------
```

```
--
-- Name       : LDAP_ATTR
-- Data Type  : OBJECT
-- DESCRIPTION : This structure contains details regarding
--              an attribute.
--
-----------------------------------------------------------------
CREATE TYPE LDAP_ATTR AS OBJECT (
     attr_name       VARCHAR2(255),
     attr_value      VARCHAR2(2048),
     attr_bvalue     RAW(2048),
     attr_value_len  INTEGER,
     attr_type       INTEGER  -- (0 - String, 1 - Binary)
     attr_mod_op     INTEGER
);
/
 GRANT EXECUTE ON LDAP_ATTR to public;


-----------------------------------------------------------
--
-- Name       : LDAP_ATTR_LIST
-- Data Type  : COLLECTION
-- DESCRIPTION : This structure contains collection
--              of attributes.
--
-----------------------------------------------------------
CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
 GRANT EXECUTE ON LDAP_ATTR_LIST to public;


---------------------------------------------------------------------------
--
-- NAME       : LDAP_NTFY
-- DESCRIPTION : This is a notifier interface implemented by Provisioning System
--              clients to receive information about changes in Oracle Internet
--              Directory. The name of package can be customized as needed.
--              The function names within this package should not be changed.
--
--
---------------------------------------------------------------------------
CREATE OR REPLACE PACKAGE LDAP_NTFY AS


--
-- LDAP_NTFY data type definitions
--


-- Event Types
USER_DELETE              CONSTANT VARCHAR2(256) := 'USER_DELETE';
USER_MODIFY              CONSTANT VARCHAR2(256) := 'USER_MODIFY';
GROUP_DELETE             CONSTANT VARCHAR2(256) := 'GROUP_DELETE';
GROUP_MODIFY             CONSTANT VARCHAR2(256) := 'GROUP_MODIFY';

-- Return Codes (Boolean)
SUCCESS                  CONSTANT NUMBER  := 1;
FAILURE                  CONSTANT NUMBER  := 0;

-- Values for attr_mod_op in LDAP_ATTR object.
MOD_ADD                  CONSTANT NUMBER  := 0;
MOD_DELETE               CONSTANT NUMBER  := 1;
MOD_REPLACE              CONSTANT NUMBER  := 2;
```

ORACLE®

```
--------------------------------------------------------------------------------
------------------
-- Name: LDAP_NTFY
-- DESCRIPTION: This is the interface to be implemented by Provisioning System
--              clients to send information to and receive information from
--              Oracle Internet Directory. The name of the package can be
--              customized as needed. The function names within this package
--              should not be changed.

--------------------------------------------------------------------------------
------------------

CREATE OR REPLACE PACKAGE LDAP_NTFY AS
```

13.6.2 Predefined Event Types

Predefined Event Types include ADD, DELETE, MODIFY functions.

```
ENTRY_ADD               CONSTANT VARCHAR2 (32)    := 'ENTRY_ADD';
ENTRY_DELETE            CONSTANT VARCHAR2 (32)    := 'ENTRY_DELETE';
ENTRY_MODIFY            CONSTANT VARCHAR2 (32)    := 'ENTRY_MODIFY';

USER_ADD                CONSTANT VARCHAR2 (32)    := 'USER_ADD';
USER_DELETE             CONSTANT VARCHAR2 (32)    := 'USER_DELETE';
USER_MODIFY CONSTANT    VARCHAR2(32)              := 'USER_MODIFY';

IDENTITY_ADD            CONSTANT VARCHAR2 (32)    := 'IDENTITY_ADD';
IDENTITY_DELETE         CONSTANT VARCHAR2 (32)    := 'IDENTITY_DELETE';
IDENTITY_MODIFY         CONSTANT VARCHAR2 (32)    := 'IDENTITY_MODIFY';

GROUP_ADD               CONSTANT VARCHAR2 (32)    := 'GROUP_ADD';
GROUP_DELETE            CONSTANT VARCHAR2 (32)    := 'GROUP_DELETE';
GROUP_MODIFY            CONSTANT VARCHAR2 (32)    := 'GROUP_MODIFY';

SUBSCRIPTION_ADD        CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_ADD';
SUBSCRIPTION_DELETE     CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_DELETE';
SUBSCRIPTION_MODI       CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_MODIFY';

SUBSCRIBER_ADD          CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_ADD';
SUBSCRIBER_DELETE       CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_DELETE';
SUBSCRIBER_MODIFY       CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_MODIFY';
```

13.6.3 Attribute Type

Attribute Type include string, binary and encrypted string attributes.

```
ATTR_TYPE_STRING            CONSTANT NUMBER    := 0;
ATTR_TYPE_BINARY            CONSTANT NUMBER    := 1;
ATTR_TYPE_ENCRYPTED_STRING  CONSTANT NUMBER    := 2;
```

13.6.4 Attribute Modification Type

Modification Type include MOD_ADD, MOD_DELETE and MOD_REPLACE attributes.

```
MOD_ADD       CONSTANT NUMBER    := 0;
MOD_DELETE    CONSTANT NUMBER    := 1;
MOD_REPLACE   CONSTANT NUMBER    := 2;
```

13.6.5 Event Dispositions Constants

Event dispositions constants include EVENT_SUCCESS, EVENT_ERROR, and EVENT_RESEND.

```
EVENT_SUCCESS      CONSTANT VARCHAR2(32)      := 'EVENT_SUCCESS';
EVENT_ERROR        CONSTANT VARCHAR2(32)      := 'EVENT_ERROR';
EVENT_RESEND       CONSTANT VARCHAR2(32)      := 'EVENT_RESEND';
```

13.6.6 About Callbacks

A callback is a function invoked by the provisioning integration service to send or receive notification events. While transferring events for an object, the related attributes can also be sent along with other details. The attributes are delivered as a collection (array) of attribute containers, which are in unnormalized form: if an attribute has two values, two rows are sent in the collection.

This section contains the following topics:

- About GetAppEvent() API
- About PutAppEventStatus() API
- About PutOIDEvent() API

13.6.6.1 About GetAppEvent() API

The Oracle Directory Integration and Provisioning server invokes this API in the remote database. It is up to the application to respond with an event. The Oracle Directory Integration and Provisioning processes the event and sends the status back using the `PutAppEventStatus()` callback. The return value of `GetAppEvent()` indicates whether an event is returned or not.

```
FUNCTION GetAppEvent (event OUT LDAP_EVENT)
RETURN NUMBER;

-- Return CONSTANTS
EVENT_FOUND           CONSTANT NUMBER  := 0;
EVENT_NOT_FOUND       CONSTANT NUMBER  := 1403;
```

If the provisioning server is not able to process the event—that is, it runs into some type of LDAP error—it responds with `EVENT_RESEND`. The application is expected to resend that event when `GetAppEvent()` is invoked again.

If the provisioning server is able to process the event, but finds that the event cannot be processed—for example, the user to be modified does not exist, or the user to be subscribed does not exist, or the user to be deleted does not exist—then it responds with `EVENT_ERROR` to indicate to the application that something was wrong. Resending the event is not required. It is up to the application to handle the event.

Note the difference between `EVENT_RESEND` and `EVENT_ERROR` in the previous discussion. `EVENT_RESEND` means that it was possible to apply the event but the server could not. If it gets the event again, it might succeed.

`EVENT_ERROR` means there is no error in performing directory operations, but the event could not be processed due to other reasons.

13.6.6.2 About PutAppEventStatus() API

The Oracle Directory Integration and Provisioning server invokes this callback in the remote database after processing an event it has received using the `GetAppEvent()` callback. For every event received, the Oracle Directory Integration and Provisioning server sends the status event back after processing the event.

```
PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS);
```

13.6.6.3 About PutOIDEvent() API

The Oracle Directory Integration and Provisioning server invokes this API in the remote database. It sends event to applications using this callback. It also expects a status event object in response as an `OUT` parameter. If a valid event status object is not sent back, or it indicates a `RESEND`, the Oracle Directory Integration and Provisioning server resends the event. In case of `EVENT_ERROR`, the server does not resend the event.

```
PROCEDURE PutOIDEvent (event  IN  LDAP_EVENT,   event_status  OUT
LDAP_EVENT_STATUS);
END LDAP_NTFY;
/
```

Part III

Appendixes

The plug-ins are used to customize provisioning in Oracle Collaboration Suite, DSML syntax and usage and Migrating from Netscape LDAP SDK API to Oracle LDAP SDK API are documented in this Appendix.

- Java Plug-ins for User Provisioning
- DSML Syntax
- Migrating from Netscape LDAP SDK API to Oracle LDAP SDK API

A
Java Plug-ins for User Provisioning

The Oracle provisioning server cannot support all of the provisioning needs of a deployment. Hence, hooks are provided at various stages of user creation, modification, and deletion. These hooks enable an enterprise to incorporate its own business rules and to tailor information creation to its needs. The hooks take the form of Java plug-ins.

The following topics describe how to use Java plug-ins to customize provisioning policy evaluation, data validation, data manipulation, and event delivery in typical deployments of Oracle Directory Integration and Provisioning Service version 3.0:

- Understanding Plug-in Types Used in Provisioning
- Understanding the Requirements for Provisioning Plug-ins
- Data Entry Provisioning Plug-ins
- Implementing Data Access Provisioning Plug-in
- Implementing Event Delivery Provisioning Plug-in
- Return Status for Provisioning Plug-ins
- Configuration Template to Implement Provisioning Plug-ins
- Provisioning Plug-in Sample Code

A.1 Understanding Plug-in Types Used in Provisioning

Plug-ins are used in provisioning to achieve data entry, data provisioning and event delivery.

There are three types of provisioning plug-ins:

- Data entry plug-ins
- Data manipulation and data access plug-ins
- Event Delivery plug-ins

The data entry plug-ins can be used by applications that integrate with the provisioning framework using either synchronous or asynchronous provisioning. The data access plug-ins are used only by applications that are integrated with the provisioning framework for synchronous provisioning. The event delivery plug-ins are used only by applications that integrate with the provisioning framework using asynchronous provisioning.

Oracle Provisioning Console, Oracle Directory Integration and Provisioning server, and other mechanisms that affect the base user information in the directory invoke these plug-ins when the information is created. By configuring a data entry plug-in, a deployment can do any of the following:

- Validate attribute values for application users
- Validate attribute values for base users
- Enhance attribute values for application users
- Enhance attribute values for base users

- Evaluate provisioning policies

If you want the deployed application to maintain application user information you must configure a data access plug-in for it. This type of plug-in enables you to maintain the application information either outside of the directory or within it as several entries.

Data entry and data access plug-ins are typically invoked from one of these environments:

- User provisioning console for Oracle Delegated Administration Services
- Oracle Directory Integration and Provisioning server
- Provisioning API
- Bulk Provisioning Tools

The event delivery plug-ins are required by applications that have the JAVA interface type and that subscribe for provisioning events. Applications that have synchronous provisioning should not implement event delivery plug-ins.

A.2 Understanding the Requirements for Provisioning Plug-ins

All of the plug-ins that you provide for an application must be in a JAR file that can be uploaded to the directory with the standard LDIF template.

See the section Configuration Template to Implement Provisioning Plug-ins for an example. The plug-in interface definitions are found in `$ORACLE_HOME/jlib/ldapjclnt10.jar`. Refer to *Oracle Fusion Middleware Java API Reference for Oracle Internet Directory* and the public interfaces for a more detailed description. If the application requires additional jar files, you can upload them too.

Place the files in the directory:

```
$ORACLE_HOME/user_projects/domains/DOMAIN_NAME/servers/
MANAGED_SERVER_NAME/tmp/_WL_user/DIP_VERSION_NUMBER/RANDOM_CHARACTERS/APP-
INF/lib/
```

A.3 Data Entry Provisioning Plug-ins

To use Data entry provisioning plug-ins you need to implement `oracle.idm.provisioning.plugin.IdataEntryPlugin` interface.

Data entry plug-ins take two forms:

- Pre–data-entry plug-ins
- Post–data-entry plug-ins

This section contains the following topics:

- Implementing Data Entry Provisioning Plug-ins
- About Application Context Object
- Understanding the Calling Sequence
- Using Pre-Data-Entry Provisioning Plug-in

A.3.1 Implementing Data Entry Provisioning Plug-ins

You use `oracle.idm.provisioning.plugin.IdataEntryPlugin` interface to provision plug-ins.

This interface has three methods, which are described as follows:

```
/**
   * The applications can perform a post data entry operation by
   * implementing this method.
   *
   * @param appCtx the application context
   * @param idmUser the IdmUser object
   * @param baseUserAttr Base user properties
   * @param appUserAttr App user properties
   * @throws PluginException when an exception occurs.
   */
 public PluginStatus process(ApplicationContext appCtx,
    IdmUser idmUser, ModPropertySet baseUserAttr,
      ModPropertySet appUserAttr)throws PluginException;
 /**
   * Returns the Modified Base User properties
   *
   * @return ModPropertySet modified base user properties.
   */
 public ModPropertySet getBaseAttrMods();

 /**
   * Returns the Modified App User properties
   *
   * @return ModPropertySet modified app user properties.
   */
 public ModPropertySet getAppAttrMods();
```

Typically the plug-in implementer uses these methods for data validation or policy evaluation. In the latter case, a base user attribute is used to make the decision.

A.3.2 About Application Context Object

The application context object has the information on LDAP directory, plug-ins, callback and log methods.

The application context object information is explained below:

• LDAP directory context

If you want the application to perform a directory operation, you can have it obtain the LDAP context from the application object. Note that this LDAP context should not be closed in the plug-in.

• Plug-in call mode

The plug-in is called from Oracle Provisioning Console, Oracle Directory Integration and Provisioning server, or another environment that invokes the provisioning API. If the calling environment is Oracle Directory Integration and Provisioning, the provisioning service calls the plug-in. The two possible values are `INTERACTIVE_MODE` and `AUTOMATIC_MODE`. The first indicates that the plug-in was invoked through interaction

between Oracle Delegated Administration Services and a client application. The second indicates that the plug-in was invoked by Oracle Directory Integration and Provisioning, where user intervention does not occur.

- Client locale

  The plug-in may want to know what the client locale is, especially if it is invoked from Oracle Delegated Administration Services.

- Plug-in call operation

  You may decide to have data entry plug-ins for both create and modify user operations. You may even implement these plug-ins in the same class. Under these conditions, the plug-in must determine which operation is invoked. The application context object uses the values `OP_CREATE` and `OP_MODIFY` to identify the operation.

- Plug-in invocation point

  The data entry plug-in is typically used to determine whether a user must be provisioned for an application. The policy evaluation and data validation that occurs can be performed in either a pre–data-entry plug-in or a post–data-entry plug-in. You may choose either or both. If you choose both, you can implement them in the same class. The application context object specifies which one is actually invoked. It uses the values `PRE_DATA_ENTRY` and `POST_DATA_ENTY` to do this.

- Callback context

  If you decide to have both pre and post plug-ins for an operation and you want the pre plug-in to share information with the post plug-in, you can set the callback context in the application context object of the pre–data-entry plug-in. The post–data-entry plug-in can then obtain and use this callback context.

- Logging

  You can use the log methods provided in the application context object to log information for the plug-in.

A.3.3 Understanding the Calling Sequence

The calling sequence constructs an application context object and merges the user attributes with application user attributes.

The calling sequence looks like this:

1. Download and instantiate a plug-in object based on the configuration information object in Oracle Internet Directory

2. Construct an application context object that is passed to the plug-in.

3. Call `process method()`

4. Call `getBaseAttrMods()` to obtain base user attributes that are modified in `process()`.

5. Merge the base user attributes returned by `getBaseAttrMods()` with the base user attributes, depending on the plug-in execution status. The execution status can be either `success` or `failure`. The plug-in implementer must return a valid plug-in execution status object. If null is returned, the execution status is considered a failure.Merging of the base user is only done if the plug-in execution status is successful.

6. Call `getAppAttrMods()` for the plug-in. This method obtains application user attributes that are modified in `process()`.

7. Merge the application user attributes returned by `getAppAttrMods()` with the application user attributes, depending on the user provisioning status returned by the plug-in.

A.3.4 Using Pre-Data-Entry Provisioning Plug-in

The pre–data-entry plug-in generates values for application attributes. The attribute defaults specified during application registration are passed to this plug in along with the current base user attributes. The returned values are displayed in the UI if the invocation environment is interactive like Oracle Delegated Administration Services.

The pre–data-entry plug-in can decide whether the user should be provisioned for an application. The plug-in examines base user attributes to make the decision. It is invoked during create and modify operations. You can support both operations with one plug-in class, or you can assign one class to each.

If the application decides to have pre–data- entry plug-ins for create and modify operations, two configuration entries must be created in Oracle Internet Directory under the application container. The first entry is for the create operation:

```
dn: cn=PRE_DATA_ENTRY_CREATE, cn=Plugins, cn=FILES, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserCreatePlugin
orclODIPPluginAddInfo: Pre Data Entry Plugin for CREATE operation
```

The second entry is for the modify operation:

```
dn: cn=PRE_DATA_ENTRY_MODIFY, cn=Plugins, cn=FILES, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserModifyPlugin
orclODIPPluginAddInfo: Pre Data Entry Plugin for MODIFY operation
```

In this example, separate classes for create and modify plug-ins are shown.

A.3.5 Using Post-Data-Entry Provisioning Plug-in

The post–data-entry plug-in validates data entered by the user in the UI. In addition, it generates derived attribute values. If the plug in fails for any one application, the UI does not proceed. All applications must successfully validate the data before a user entry can be created in the directory. However, in the case of non-UI environment or automatic route, the plug-in implementer can decide to raise an error or continue, based on the plug-in call mode (`INTERACTIVE_MODE` or `AUTOMATIC_MODE`).

Like the pre–data-entry plug-in, the post–data-entry plug-in is invoked during create and modify operations. The application can decide to implement one plug-in class for both operations or a separate class for each.

If you decide to have post–data-entry plug-ins for create and modify operations, create two configuration entries in Oracle Internet Directory under the application container. The first entry is for the create operation:

```
dn: cn=POST_DATA_ENTRY_CREATE, cn=Plugins, cn=FILES, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserMgmtPlugin
orclODIPPluginAddInfo: Post Data Entry Plugin for CREATE and MODIFY
 operations
```

The second entry is for the modify operation:

```
dn: cn=POST_DATA_ENTRY_MODIFY, cn=Plugins, cn=FILES, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserMgmtPlugin
orclODIPPluginAddInfo: Post Data Entry Plugin for MODIFY and CREATE operation
```

In this example, too, separate classes for create and modify plug-ins are shown.

A.4 Implementing Data Access Provisioning Plug-in

The primary purpose of the data access plug in is to manage the application-specific information of the user in the directory. You can use this plug-in to create and retrieve the information.

The data access plug-in is invoked whenever a user is created and is requesting provisioning for an application—whether by Oracle Delegated Administration Services, by Oracle Directory Integration and Provisioning, or by bulk provisioning tools.

The data access plug-in is invoked during modify and delete operations as well. It can update the application information or remove it.

If you want to use the data access plug-in, implement the interface `oracle.idm.provisioning.plugin.IDataAccessPlugin`. Here is the interface:

```
/**
 * The applications can create/modify/delete the user footprint by
 * implementing this method.
 *
 * @param appCtx the application context
 * @param idmUser IdmUser object
 * @param baseUserAttr Base user properties
 * @param appUserAttr App user properties
 *
 * @return PluginStatus a plugin status object, which must contain
 * the either <codE>IdmUser.PROVISION_SUCCESS</CODE> or
 * <codE>IdmUser.PROVISION_FAILURE</CODE> provisioning status
 *
 * @throws PluginException when an exception occurs.
 */
public PluginStatus process(ApplicationContext appCtx,
  IdmUser idmUser, ModPropertySet baseUserAttr,
```

```
        ModPropertySet ppUserAttr) throws PluginException;

    /**
     * The applications can return their user footprint by
     * implementing this method. Use <CODE>
     * oracle.ldap.util.VarPropertySet </CODE>
     * as the return object
     *
     * <PRE>
     *  For Ex.
     *    PropertySet retPropertySet =  null;
     *    retPropertySet = new VarPropertySet();
     *
     *    //Fetch the App data and add it to retPropertySet
     *    retPropertySet.addProperty("name", "value");
     *    ..
     *    return retPropertySet;
     * </PRE>
     *
     * @throws PluginException when an exception occurs.
     */
    public PropertySet getAppUserData(ApplicationContext appCtx,
          IdmUser user, String reqAttrs[]) throws PluginException;
```

If you want to manage the user information for an application, create a plug-in configuration entry in the directory under the application container. The example that follows shows what this entry looks like:

```
dn: cn=DATA_ACCESS, cn=Plugins, cn=FILES, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: oracle.myapp.provisioning.UserDataAccPlugin
orclODIPPluginAddInfo: Data Access Plugin
```

A.5 Implementing Event Delivery Provisioning Plug-in

The primary purpose of the event delivery plug-in is to use the events notified by the Oracle Directory Integration and Provisioning server. Events are delivered to the plug-in by the Oracle Directory Integration and Provisioning server.

Based on the event type and the action to be performed in the application repository, the plug-in performs the required operations. The interface definitions for this plug-in are as follows:

```
/* $Header: IEventPlugin.java 09-jun-2005.12:45:53  *
/* Copyright (c) 2004, 2005, Oracle. All rights reserved.  */
/*
   DESCRIPTION
    All of the plug-in interfaces must extend this common interface.
   PRIVATE CLASSES
    None
   NOTES
    None
*/
package oracle.idm.provisioning.plugin;
/**
 * This is the base interface
```

```
 */
public interface IEventPlugin
{
   /**
    * The applications can perform the initialization logic in this method.
    *
    * @param Object For now it is the provisioning Profile that is passed.
    *          look at oracle.ldap.odip.engine.ProvProfile for more details.
    *
    *
    * @throws PluginException when an exception occurs.
    */
   public void initialize(Object profile) throws PluginException;
   /**
    * The applications can perform the termination logic in this method.
    *
    * @param void Provisioning Profile Object is sent.
    *              refer to oracle.ldap.odip.engine.ProvProfile for more details
    * @throws PluginException when an exception occurs.
    */
   public void terminate(Object profile) throws PluginException;
   /**
    * Set Additional Info.
    * Since we pass on the complete profile, there is no requirement to set
    * the additiona
    * @param addInfo Plugin additional info
    */
   //public void setAddInfo(Object addInfo);
}

/* $Header: IEventsFromOID.java 09-jun-2005.12:45:53  */
/* Copyright (c) 2004, 2005, Oracle. All rights reserved.  */
/*
   DESCRIPTION
   Applications interested in receiving changes from OID should
implement this   interface.
   PRIVATE CLASSES
    <None>
   NOTES
*/
package oracle.idm.provisioning.plugin;
import oracle.idm.provisioning.event.Event;
import oracle.idm.provisioning.event.EventStatus;

/**
 * Applications interested in receiving changes from OID should implement this
 * interface.  The applications register with the OID for the changes occurring
 * at OID. The DIP engine would instantiate an object of this class and invoke
 * the initialize(), sendEventsToApp(), and truncate() method in the same
 * sequence. The initialize method would provide the appropriate information
 * from the profile in the form of a java.util.Hashtable object.
 * The property names, that is, the hash table key that could be used by the
 * interface implementer is defined as constants in this interface.
 *
 * @version $Header: IEventsFromOID.java 09-jun-2005.12:45:53  $
 */
public interface IEventsFromOID extends IEventPlugin
{

   /**
    * Initialize. The application would provide any initialization logic
```

```
    * through method. The DIP engine after instantiating a class that
    * implements this interface will first invoke this method.
    *
    * @param prop A HashMap that would contain necessary information exposed
    * to the applications
    * @throws EventInitializationException the applications must throw this
    * exception in case of error.
    */
   public void initialize(Object provProfile)
                               throws EventPluginInitException;


   /**
    * OID Events are deliverd to the application through this method.
    *
    * @param evts an array of LDAPEvent objects returned by the DIP engine
    * @return the application logc must process these events and return the
    * status of the processed events
    * @throws EventDeliveryException the applications must throw this exception
    * in case of any error.
    */
   public EventStatus[] sendEventsToApp(Event [] evts)
                           throws EventDeliveryException;
}

/* $Header: IEventsToOID.java 09-jun-2005.12:45:53 $ */
/* Copyright (c) 2004, 2005, Oracle. All rights reserved.  */

/*
   DESCRIPTION
   Applications interested in sending changes to OID should implement this
   interface.
*/
package oracle.idm.provisioning.plugin;
import oracle.idm.provisioning.event.Event;
import oracle.idm.provisioning.event.EventStatus;

/**
 * Applications interested in sending changes to OID should implement this
 * interface.  The applications must register with the OID for the sending
 * changes at their  end to DIP. The DIP engine would instantiate an object
 * of this class and invoke the initialize(), sendEventsFromApp(), and
 * truncate() method in the same sequence. The initialize method would
 * provide the appropriate information from the profile in the form of
 * a java.util.Hashtable object. The property names, that is, the hash table key
 * that could be used by the interface implementer is defined as
 * constants in this interface.
 *
 */
public interface IEventsToOID extends IEventPlugin
{
   /**
    * Initialize. The application would provide any initialization logic
    * through method. The DIP engine after instantiating a class that
    * implements this interface will first invoke this method.
    *
    * @param  prop ProvProfile
    *               oracle.ldap.odip.engine.ProvProfile
    * @throws EventPluginInitException the applications must throw this
    * exception in case of error.
    */
   public void initialize(Object profile) throws EventPluginInitException;
```

```
/**
 * Application Events are deliverd to OID through this method.
 *
 * @return an array of Event objects returned to be processed by the
 * DIP engine.
 * @throws EventDeliveryException the applications must throw this exception
 * in case of any error.
 */
public Event[] receiveEventsFromApp()
                        throws EventDeliveryException;

/**
 * Application can let the DIP engine know whether there are more event to
 * follow through this method
 *
 * @return ture if there are more events to be returned and false otherwise
 * @throws PluginException the applications must throw this exception
 * in case of any error.     */
public boolean hasMore() throws PluginException;

/**
 * The status of the application events are intimated through this method.
 * i.e the DIP engine after processing the events calls this method to set
 * the event status.
 *
 * @param an array of Event status objects describing the processed event
 * status by the DIP engine.
 * @throws EventDeliveryException the applications must throw this exception
 * in case of any error.
 */
public void setAppEventStatus(EventStatus[] evtStatus)
                        throws EventDeliveryException;
}
```

To perform directory operations from a plug-in, you need the application context. You can use `ProvProfile.getApplicationContext()` in the event delivery plug-in `initialize()` method to get an instance of `oracle.idm.provisioning.plugin.ApplicationContext`.You can use this `applicationContext` to perform any directory operation in any plug-in method.

A.6 Return Status for Provisioning Plug-ins

Each of the provisioning plug-ins must return an object of the appropriate class.

IDataEntryPlugin and IDataAccessPlugins return an object of the class oracle.idm.provisioning.plugin.PluginStatus. The EventDeliveryPlugins (IEventFromOID and IEventToOID) return an array of objects of the class 'oracle.idm.provisioning.event.EventStatus'.

The returned object indicates the execution status, which is either `success` or `failure`. The object can return the user provisioning status as well.

A.7 Configuration Template to Implement Provisioning Plug-ins

The LDIF template provided here is used in Oracle Internet Directory to specify the application plug-in.

You must create a directory entry for the application and upload the JAR file that contains the classes that implement the plug-in.

```
dn: cn=Plugins, cn=APPTYPE, cn=Applications, cn=Provisioning,
 cn=Directory Integration Platform,cn=Products,cn=OracleContext
changetype: add
add: orclODIPPluginExecData
orclODIPPluginExecData: full_path_name_of_the_JAR_file
objectclass: orclODIPPluginContainer

dn: cn=PRE_DATA_ENTRY_CREATE, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning, cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Pre Data Entry Plugin for CREATE operation

dn: cn=PRE_DATA_ENTRY_MODIFY, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Pre Data Entry Plugin for MODIFY operation

dn: cn=POST_DATA_ENTRY_CREATE, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Post Data Entry Plugin for CREATE and modify operations

dn: cn=POST_DATA_ENTRY_MODIFY, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Post Data Entry Plugin for MODIFY and CREATE operation

dn: cn=DATA_ACCESS, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning,  cn=Directory Integration Platform, cn=Products,
 cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
```

```
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Data Access Plugin

dn: cn=EVENT_DELIVERY_OUT, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning, cn=Directory Integration Platform, cn=Products,
cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Event Delivery Plugin for Outbound

dn: cn=EVENT_DELIVERY_IN, cn=Plugins, cn=APPTYPE, cn=Applications,
 cn=Provisioning, cn=Directory Integration Platform, cn=Products,
cn=OracleContext
changetype: add
objectClass: orclODIPPlugin
orclStatus: ENABLE
orclODIPPluginExecName: Name_of_the_class_that_implements_the_plug-in
orclODIPPluginAddInfo: Event Delivery Plugin for Inbound
```

A.8 Provisioning Plug-in Sample Code

Pre data entry plugin for CREATE operation that validates the attribute is described.

The following sample code describes provisioning plug-in:

```
/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
 DESCRIPTION
 Sample PRE DATA Entry Plugin for CREATE operation that
 validates the attribute.
 PRIVATE CLASSES
 None.
 NOTES
 This class implements the PRE_DATA_ENTRY_CREATE plugin ONLY
 MODIFIED (MM/DD/YY)
 12/15/04 \226 Creation
*/
package oracle.ldap.idm;

import java.util.*;
import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
/**
 * This class implements the PRE_DATA_ENTRY_CREATE plugin ONLY
 *
 */
public class SamplePreDataEntryCreatePlugin implements IDataEntryPlugin
{
   public ModPropertySet mpBaseUser = null;
   public ModPropertySet mpAppUser = null;

   public PluginStatus process(ApplicationContext appCtx,IdmUser idmuser,
    ModPropertySet baseUserAttr, ModPropertySet appUserAttr)
         throws PluginException
   {
```

```
        PluginStatus retPluginStatus = null;
        String retProvStatus = null;
        String retProvStatusMsg = null;

        LDIFRecord lRec = null;
        LDIFAttribute lAttr = null;
        String val = null;
        if(null == baseUserAttr.getModPropertyValue(\223departmentNumber\224))
        {
           mpBaseUser = new ModPropertySet();
           mpBaseUser.addProperty("departmentNumber","ST");
           appCtx.log(\223Base user attribute \226 departmentNumber missing\224 +
                   \223Setting default - ST\224);
        }
        else if ( baseUserAttr.getModPropertyValue(\223departmentNumber\224)
                   .notIn(\223ST\224, \223APPS\224, \224CRM\224) )
        {
           throw new PluginException(\223Invalid department Number\224);
        }
        if((null == appUserAttr) ||
                   null == appUserAttr.getModPropertyValue(\223emailQouta\224))
        {
           mpAppUser = new ModPropertySet();
           mpAppUser.addProperty("emailQouta","50M");
           appCtx.log(\223Application user attribute - email Qouta missing \224 +
               \223Setting default - 50M\224);
        }
        return new PluginStatus(PluginStatus.SUCCESS, null, null);
    }

  public ModPropertySet getBaseAttrMods()
  {
    return mpBaseUser;
  }

  public ModPropertySet getAppAttrMods()
  {
    return mpAppUser;
  }
}

/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
 DESCRIPTION
 Sample POST DATA Entry Plugin for CREATE operation. Implementing a
 policy check to provision only those users who belong to \223SALES\224.
 PRIVATE CLASSES
 None.
 NOTES
 This class implements the POST_DATA_ENTRY_CREATE plugin ONLY
 MODIFIED (MM/DD/YY)
 12/15/04 \226 Creation
*/
package oracle.ldap.idm;

import java.util.*;
import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
```

```
/**
 * This class implements the POST_DATA_ENTRY_CREATE plugin ONLY
 *
 */
public class SamplePostDataEntryCreatePlugin implements IDataEntryPlugin
{
    public ModPropertySet mpBaseUser = null;
    public ModPropertySet mpAppUser = null;

    public PluginStatus process(ApplicationContext appCtx,IdmUser idmuser,
     ModPropertySet baseUserAttr, ModPropertySet appUserAttr)
            throws PluginException
  {
    PluginStatus retPluginStatus = null;
    String retProvStatus = null;
    String retProvStatusMsg = null;

    if(null == baseUserAttr.getModPropertyValue(\223deptartmentNumber\224))
    {
       mpBaseUser = new ModPropertySet();
       mpBaseUser.addProperty("deptartmentNumber ","SALES");
       appCtx.log("Base user attribute \221c\222 is missing");

       retProvStatus = IdmUser.PROVISION_ REQUIRED;
       retProvStatusMsg =  "Provision policy: Only \221SALES\222\224.
    }
    else if (baseUserAttr.getModPropertyValue(\223deptartmentNumber\224)
            .equals(\223SALES\224))
    {
       retProvStatus = IdmUser.PROVISION_ REQUIRED;
       retProvStatusMsg =  "Provision policy: Only \221SALES\222\224.
    }
    else
    {
       // do not provision those users who do not belong to SALES.
       retProvStatus = IdmUser.PROVISION_NOT_REQUIRED;
       retProvStatusMsg =
               "Do not provision the person who is not from \221SALES\222";
     }

     return new PluginStatus(PluginStatus. SUCCESS, retProvStatusMsg,
                                 retProvStatus);
  }

  public ModPropertySet getBaseAttrMods()
  {
    return mpBaseUser;
  }

  public ModPropertySet getAppAttrMods()
  {
    return mpAppUser;
  }
}

/* Copyright (c) 2004, Oracle. All rights reserved. */
/**
 DESCRIPTION
 Sample DATA Access Plugin.
 NOTES
 This class implements the DATA_ACCESS plugin
```

```
 MODIFIED (MM/DD/YY)
 12/15/04 \226 Creation
*/
package oracle.ldap.idm;

import javax.naming.*;
import javax.naming.ldap.*;
import javax.naming.directory.*;
import oracle.ldap.util.*;
import oracle.idm.provisioning.plugin.*;
/**
 * This class implements the DATA_ACCESS plugin ONLY
 *
 */
public class SampleDataAccessPlugin implements IDataAccessPlugin
{
   public PluginStatus process(ApplicationContext appCtx,IdmUser idmuser,
        ModPropertySet baseUserAttr,ModPropertySet appUserAttr)
            throws PluginException
   {
      try {
          DirContext dirCtx = appCtx.getDirCtx();
          if ( appCtx.getCallOp().equals(ApplicationContext.OP_CREATE )
          {
            // Use the directory context and create the entry.
          }
          elseif ( appCtx.getCallOp().equals(ApplicationContext.OP_MODIFY)
          {
            // Use the directory context and modify the entry.
          }
      } catch (Exception e) {
          throw new PluginException(e);
      }
      return new PluginStatus(PluginStatus.SUCCESS, null, null);
   }

   public PropertySet getAppUserData(ApplicationContext appCtx,
          IdmUser idmuser, String [] reqAttrs) throws PluginException
   {
      VarPropertySet vpSet = null;
      DirContext dirCtx = appCtx.getDirCtx();

      try {
        Attributes attrs= dirCtx.getAttributes(\223myAppContainer\224);
        vpSet = new VarPropertySet(); // Populate the VarPropertySet from attrs
      } catch(Exception ne) {
        throw new PluginException(e);
      }
      return vpSet;    }
}
```

B

DSML Syntax

Directory Services Mark-up Language (DSML) defines the XML representation of LDAP information and operations.
The following topics describe syntax for the Directory Services Mark-up Language (DSML):

- Capabilities of DSML
- Benefits of DSML
- Understanding DSML Syntax
- Tools Enabled for DSML

> **Note:**
>
> DSML is deprecated in Oracle Fusion Middleware 11g Release 1 (11.1.1) and might not be supported in future releases.

B.1 Capabilities of DSML

Directory services form a core part of distributed computing. XML is becoming the standard markup language for Internet applications. As directory services are brought to the Internet, there is a pressing and urgent need to express the directory information as XML data. This caters to the growing breed of applications that are not LDAP-aware yet require information exchange with a LDAP directory server.

Directory Services Mark-up Language (DSML) defines the XML representation of LDAP information and operations. The LDAP Data Interchange Format (LDIF) is used to convey directory information, or a set of changes to be applied to directory entries. The former is called Attribute Value Record and the latter is called Change Record.

B.2 Benefits of DSML

Using DSML with Oracle Internet Directory and Internet applications makes it easier to flexibly integrate data from disparate sources. Also, DSML enables applications that do not use LDAP to communicate with LDAP-based applications, easily operating on data generated by an Oracle Internet Directory client tool or accessing the directory through a firewall.

DSML is based on XML, which is optimized for delivery over the Web. Structured data in XML is uniform and independent of application or vendors, thus making possible numerous new flat file type synchronization connectors. After it is in XML format, the directory data can be made available in the middle tier and have more meaningful searches performed on it.

B.3 Understanding DSML Syntax

A DSML version 1 document describes either directory entries, a directory schema or both. Each directory entry has a unique name called a distinguished name (DN).

A directory entry has several property-value pairs called directory attributes. Every directory entry is a member of several object classes. An entry's object classes constrain the directory attributes the entry can take. Such constraints are described in a directory schema, which may be included in the same DSML document or may be in a separate document.

The following subsections briefly explain the top-level structure of DSML and how to represent the directory and schema entries:

- DSML Top-Level Structure
- Directory Entries in DSML
- Schema Entries in DSML

B.3.1 DSML Top-Level Structure

The top-level document element of DSML is of the type `dsml`, which has child elements `directory-entries` and `directory-schema` elements.

The child element directory-entries may in turn have child elements of the type entry. Similarly the child element directory-schema may in turn have child elements of the types class and attribute-type.

At the top level, the structure of a DSML document looks like this:

```
<!- a document with directory & schema entries -->
  <dsml:directory-entries>
    <dsml:entry dn="...">...</dsml:entry>
    .
    .
    .
  </dsml:directory-entries>
  .
  .
  .
  <dsml:directory-schema>
    <dsml:class id="..." ...>...</dsml:class>
    <dsml:attribute-type id="..." ...>...</dsml:attribute-type>
    .
    .
    .
  </dsml:directory-schema> </dsml:dsml>
```

B.3.2 Directory Entries in DSML

The element type `entry` represents a directory entry in a DSML document. The `entry` element contains elements representing the entry's directory attributes. The distinguished name of the entry is indicated by the XML attribute `dn`.

Here is an XML entry to describe the directory entry:

```
<dsml:entry dn="uid=Heman, c=in, dc=oracle, dc=com">
<dsml:objectclass>
  <dsml:oc-value>top</dsml:oc-value>
  <dsml:oc-value ref="#person">person</dsml:oc-value>
  <dsml:oc-value>organizationalPerson</dsml:oc-value>
  <dsml:oc-value>inetOrgPerson</dsml:oc-value>
</dsml:objectclass>
<dsml:attr name="sn">
<dsml:value>Siva</dsml:value></dsml:attr>
<dsml:attr name="uid">
<dsml:value>Heman</dsml:value></dsml:attr>
<dsml:attr name="mail">
<dsml:attr name="givenname">
<dsml:value>Siva V. Kumar</dsml:value></dsml:attr>
<dsml:attr name="cn">
<dsml:value>SVK@example.com</dsml:value></dsml:attr>
<dsml:value>Siva Kumar</dsml:value></dsml:attr>
```

The `oc-value`'s `ref` is a URI Reference to a class element that defines the object class. In this case it is a URI [9] Reference to the element that defines the `person` object class. The child elements `objectclass` and `attr` are used to specify the object classes and the attributes of a directory entry.

B.3.3 Schema Entries in DSML

The element type `class` represents a schema entry in a DSML document. The `class` element takes an XML attribute `id` to make referencing easier.

For example, the object class definition for the `person` object class might look like the following:

```
<dsml:class id="person" superior="#top"  type="structural">
  <dsml:name>person</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.6.6</object-identifier>
  <dsml:attribute ref="#sn" required="true"/>
  <dsml:attribute ref="#cn" required="true"/>
  <dsml:attribute ref="#userPassword" required="false"/>
  <dsml:attribute ref="#telephoneNumber" required="false"/>
  <dsml:attribute ref="#seeAlso" required="false"/>
  <dsml:attribute ref="#description" required="false"/>
</dsml:class>
```

The directory attributes are described in a similar way. For example, the attribute definition for the `cn` attribute may look like this:

```
<dsml:attribute-type id="cn">
  <dsml:name>cn</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.4.3</object-identifier>
  <dsml:syntax>1.3.6.1.4.1.1466.115.121.1.44</dsml:syntax>
</dsml:attribute-type>
```

B.4 Tools Enabled for DSML

With the XML framework, you can now use non-ldap applications to access directory data.

The XML framework broadly defines the access points and provides the following tools:

- `ldapadd`

- `ldapaddmt`

- `ldapsearch`

> **✎ See Also:**
>
> Oracle Internet Directory Server Administration Tools in *Oracle Fusion Middleware Reference for Oracle Identity Management* for information about syntax and usage.

The client tool `ldifwrite` generates directory data and schema LDIF files. If you convert these LDIF files to XML, you can store the XML file on an application server and query it. The query and response time is small compared to performing an LDAP operation against an LDAP server.

C

Migrating from Netscape LDAP SDK API to Oracle LDAP SDK API

You need to know the differences between the Netscape LDAP SDK and the Oracle Internet Directory LDAP SDK that are important when migrating code.
The following topics describe the differences between the Netscape LDAP SDK and the Oracle Internet Directory LDAP SDK:

- Features Comparison
- Functions Comparison
- Macros Comparison

> **Note:**
>
> The Oracle Internet Directory SDK C API is described in C API Reference .

C.1 Features Comparison

Features of Oracle Internet Directory LDAP SDK and Netscape's SDK are discussed.

The following features of the Oracle Internet Directory LDAP SDK are different from Netscape's SDK.

- In the Netscape SDK, a client must register an LDAP Rebind Call Back to handle a referral. This is automatically handled in the Oracle LDAP SDK.

- Access to the LDAP Structure is different. The LDAP handle in Netscape LDAP SDK is type opaque. Accessory functions are required to access individual fields within this handle. In the Oracle Internet Directory LDAP SDK, the LDAP structure is exposed and a client can modify individual fields within the structure.

- Use ldap_open() instead of ldap_init() with the Oracle LDAP SDK.

- SSL connection initialization requires different function calls and procedures in the Oracle LDAP SDK. See C API Reference for information about Oracle Internet Directory function calls for SSL.

- The Oracle Internet Directory C API depends on the Oracle environment, including libraries and other files.You must install Oracle Application Server or Oracle Database and set the environment variable $ORACLE_HOME to an appropriate location before you build your application.

- An LDAP SDK user must use an allocation function that clears memory, such as calloc(), to allocate an LDAPMod structure().

- The Oracle Internet Directory API is not thread-safe.

C.2 Functions Comparison

The functions that are available in Netscape LDAP SDK and not in Oracle LDAP SDK are discussed.

- The Oracle LDAP SDK does not have the function ldap_ber_free(). Use ber_free() instead.

- The Oracle LDAP SDK does not have the function ldap_get_lderrno() for retrieving the ld error and matched string. You can retrieve this information directly by accessing the field LDAP.ld_matched and LDAP.ld_error. These are the only fields of the LDAP structure that you should ever need to access.

C.3 Macros Comparison

Macros used in Oracle Internet Directory is compared in the following section.

- LDAPS_PORT is not defined in the Oracle LDAP SDK. Use LDAP_SSL_PORT instead.

- LDAP_AFFECT_MULTIPLE_DSA is not defined in the Oracle LDAP SDK. This is a Netscape-specific macro.