IBM Security Directory Server
Version 6.3.1.5

Programming Reference

SC27-2754-02

IBM Security Directory Server
Version 6.3.1.5

Programming Reference

SC27-2754-02

Note

Before using this information and the product it supports, read the general information under "Notices" on page 315.

Edition notice

Note: This edition applies to version 6.3.1.5 of *IBM Security Directory Server* (product number 5724-J39) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

IBM® Security Directory Server, previously known as IBM Tivoli® Directory Server, is an IBM implementation of Lightweight Directory Access Protocol for the following operating systems:

- Microsoft Windows
- AIX®
- Linux (System x®, System z®, System p®, and System i®)
- Solaris
- Hewlett-Packard UNIX (HP-UX) (Itanium)

IBM Security Directory Server Programming Reference contains information about writing LDAP client applications for your IBM Security Directory Server.

The book also includes:

- Sample LDAP client programs
- Sample Makefile

Access to publications and terminology

This section provides:

- A list of publications in the “IBM Security Directory Server library.”
- Links to “Online publications” on page viii.
- A link to the “IBM Terminology website” on page viii.

IBM Security Directory Server library

The following documents are available in the IBM Security Directory Server library:

- *IBM Security Directory Server, Version 6.3.1.5 Product Overview*, GC27-6212-01

  Provides information about the IBM Security Directory Server product, new features in the current release, and system requirements information.
- *IBM Security Directory Server, Version 6.3.1.5 Quick Start Guide*, GI11-9351-02

  Provides help for getting started with IBM Security Directory Server. Includes a short product description and architecture diagram, and a pointer to the product documentation website and installation instructions.
- *IBM Security Directory Server, Version 6.3.1.5 Installation and Configuration Guide*, SC27-2747-02

  Contains complete information for installing, configuring, and uninstalling IBM Security Directory Server. Includes information about upgrading from a previous version of IBM Security Directory Server.
- *IBM Security Directory Server, Version 6.3.1.5 Administration Guide*, SC27-2749-02

  Contains instructions for administrative tasks through the Web Administration tool and the command line.
- *IBM Security Directory Server, Version 6.3.1.5 Reporting Guide*, SC27-6531-00

  Describes the tools and software for creating reports for IBM Security Directory Server.

- *IBM Security Directory Server, Version 6.3.1.5 Command Reference*, SC27-2753-02

  Describes the syntax and usage of the command-line utilities included with IBM Security Directory Server.
- *IBM Security Directory Server, Version 6.3.1.5 Server Plug-ins Reference* , SC27-2750-02

  Contains information about writing server plug-ins.
- *IBM Security Directory Server, Version 6.3.1.5 Programming Reference*, SC27-2754-02

  Contains information about writing Lightweight Directory Access Protocol (LDAP) client applications in C and Java™.
- *IBM Security Directory Server, Version 6.3.1.5 Performance Tuning and Capacity Planning Guide*, SC27-2748-02

  Contains information about tuning the directory server for better performance. Describes disk requirements and other hardware requirements for directories of different sizes and with various read and write rates. Describes known working scenarios for each of these levels of directory and the disk and memory used; also suggests rules of thumb.
- *IBM Security Directory Server, Version 6.3.1.5 Troubleshooting Guide*, GC27-2752-02

  Contains information about possible problems and corrective actions that can be taken before you contact IBM Software Support.
- *IBM Security Directory Server, Version 6.3.1.5 Error Message Reference*, GC27-2751-02

  Contains a list of all warning and error messages associated with IBM Security Directory Server.

Online publications

IBM posts product publications when the product is released and when the publications are updated at the following locations:

IBM Security Directory Server documentation website

The http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDS.doc/welcome.htm site displays the documentation welcome page for this product.

IBM Security Systems Documentation Central and Welcome page

IBM Security Systems Documentation Central provides an alphabetical list of all IBM Security Systems product documentation. You can also find links to the product documentation for specific versions of each product.

Welcome to IBM Security Systems documentation provides and introduction to, links to, and general information about IBM Security Systems documentation.

IBM Publications Center

The http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss site offers customized search functions to help you find all the IBM publications you need.

IBM Terminology website

The IBM Terminology website consolidates terminology for product libraries in one location. You can access the Terminology website at http://www.ibm.com/software/globalization/terminology.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For more information, see the Accessibility Appendix in the *IBM Security Directory Server Product Overview*.

Technical training

For technical training information, see the following IBM Education website at http://www.ibm.com/software/tivoli/education.

Support information

IBM Support assists with code-related problems and routine, short duration installation or usage questions. You can directly access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html.

IBM Security Directory Server Troubleshooting Guide provides details about:

- What information to collect before you contact IBM Support.
- The various methods for contacting IBM Support.
- How to use IBM Support Assistant.
- Instructions and problem-determination resources to isolate and fix the problem yourself.

Note: The **Community and Support** tab on the product information center can provide additional support resources.

Statement of Good Security Practices

IT system security involves protecting systems and information through prevention, detection, and response to improper access from within and outside your enterprise. Improper access can result in information being altered, destroyed, misappropriated, or misused or can result in damage to or misuse of your systems, including for use in attacks on others. No IT system or product should be considered completely secure and no single product, service or security measure can be completely effective in preventing improper use or access. IBM systems, products and services are designed to be part of a comprehensive security approach, which will necessarily involve additional operational procedures, and may require other systems, products or services to be most effective. IBM DOES NOT WARRANT THAT ANY SYSTEMS, PRODUCTS OR SERVICES ARE IMMUNE FROM, OR WILL MAKE YOUR ENTERPRISE IMMUNE FROM, THE MALICIOUS OR ILLEGAL CONDUCT OF ANY PARTY.

Chapter 1. Overview

The Lightweight Directory Access Protocol (LDAP) provides TCP/IP access to LDAP-compliant servers.

The *IBM Security Directory Server Programming Reference* includes various sample LDAP client programs. It also includes an LDAP client library that is used to provide application access to the LDAP servers.

For more information, see the following sections:
- “LDAP version support”
- “LDAP API overview” on page 2

LDAP version support

The IBM Security Directory Server C-Client SDK provides support for both LDAP Version 2 and LDAP Version 3 application programming interfaces (APIs) and protocols.

The LDAP SDK APIs are based upon the "C LDAP Application Program Interface" Internet Draft at http://www.ietf.org/proceedings/45/I-D/draft-ietf-ldapext-ldap-c-api-03.txt.

The LDAP API provides typical directory functions such as read, write, and search. With the advent of support for LDAP Version 3 APIs and protocols, the following features are also supported:
- LDAP V3 referrals and search references.
- Improved globalization with UTF-8 support for Distinguished Names (DNs) and strings that are passed into, and returned from, the LDAP APIs. Support for converting string data between the local code page and UTF-8 is also provided. When you run as an LDAP V2 application, DNs and strings remain limited to the IA5 character set.
- As provided by the IBM Directory Server dynamic schema capability, an LDAP application can add, modify, and change elements of the schema. For more information, see Appendix B, “LDAP V3 schema,” on page 181.
- Controls for the LDAP server and client.

With the C-Client SDK, an application that uses the ldap_open API defaults to the LDAP V2 protocol. Existing LDAP applications continue to work and can interoperate with both LDAP V2 servers and LDAP V3 servers.

An application that uses the ldap_init API defaults to the LDAP V3 protocol with optional bind. An LDAP V3 application does not necessarily interoperate with an LDAP server that supports only LDAP V2 protocols.

Note: An application can use the `ldap_set_option` API to change its LDAP protocol version. This operation is done after you use `ldap_open` or `ldap_init` but before you issue a bind or any other operation that results in contacting the server.

LDAP API overview

The set of LDAP APIs is designed to provide a suite of functions that can be used to develop directory-enabled applications.

Directory-enabled applications typically connect to one or more directories and run various directory-related operations, such as:
- Adding entries
- Searching the directories and obtaining the resulting list of entries
- Deleting entries
- Modifying entries
- Renaming entries

The type of information that is managed in the directory depends on the nature of the application. Directories often are used to provide public access to information about people. For example:
- Phone numbers
- Email addresses
- Fax numbers
- Mailing addresses

Increasingly, directories are being used to manage and publish other types of information. For example:
- Configuration information
- Public key certificates (managed by certificate authorities (CAs))
- Access control information
- Locating information (how to find a service)

The LDAP API provides for both synchronous and asynchronous access to a directory. Asynchronous access enables your application to do other work while you wait for the results of a directory operation to be returned by the server.

Source code, example makefile, and executable programs are provided for running the following operations:
- **`ldapsearch`** (searches the directory)
- **`ldapmodify`** (modifies information in the directory)
- **`ldapdelete`** (deletes information from the directory)
- **`ldapmodrdn`** (modifies the Relative Distinguished Name (RDN) of an entry in the directory)

LDAP APIs use standard LDAP structures to store various information that is related to an LDAP operation. For information about LDAP structures, see IETF RFC 1823 at http://www.ietf.org/rfc/rfc1823.txt for version 2 of the LDAP protocol. Also, see C LDAP Application Program Interface at http://www.ietf.org/proceedings/01mar/I-D/ldapext-ldap-c-api-05.txt for version 3 of the LDAP protocol.

See *IBM Security Directory Server Command Reference*, to know more about the syntax and usage of the command-line utilities.

Typical API usage

The typical API usage provides an understanding about the LDAP connection.

The basic interaction is as follows:

1. A connection is made to an LDAP server by calling either `ldap_init` or `ldap_ssl_init`. This call is used to establish a secure connection over Secure Sockets Layer (SSL).
2. An LDAP bind operation is run by calling `ldap_simple_bind`. The bind operation is used to authenticate to the directory server. The LDAP V3 API and protocol provides the bind to be skipped, in which case the access rights associated with anonymous access are obtained.
3. Other operations are run by calling one of the synchronous or asynchronous routines. For example, `ldap_search_s` or `ldap_search` followed by `ldap_result`.
4. Results that are returned from these routines are interpreted by calling the LDAP parsing routines, which include operations such as:
   - `ldap_first_entry`, `ldap_next_entry`
   - `ldap_get_dn`
   - `ldap_first_attribute`, `ldap_next_attribute`
   - `ldap_get_values`
   - `ldap_parse_result` (new for LDAP V3)
5. The LDAP connection is terminated by calling `ldap_unbind`.

When you handle a client referral to another server, the `ldap_set_rebind_proc` routine defines the entry point of a routine that is called when an LDAP bind operation is needed.

Retrieval of results

LDAP search routines provide the accessed results.

Results that are obtained from the LDAP search routines can be accessed by calling the following APIs:

- `ldap_first_entry` and `ldap_next_entry` to step through the entries returned
- `ldap_first_attribute` and `ldap_next_attribute` to step through the attributes of an entry
- `ldap_get_values` to retrieve a value of an attribute
- `printf` or some other display or usage method

Uniform Resource Locators (URLs)

The routines support the use of LDAP URLs (Uniform Resource Locators).

Use the `ldap_url` routines to do the following actions:

- Test a URL to see whether it is an LDAP URL
- Parse LDAP URLs into their component pieces
- Initiate searches by directly using an LDAP URL

Some examples of these routines are `ldap_url_parse`, `ldap_url_search_s`, and `ldap_is_ldap_url`.

Secure Socket Layer (SSL) support

The LDAP API is extended to support connections that are protected by the SSL protocol.

This connection can be used to provide the following mechanisms:

- Strong authentication between the client and server
- Data encryption of LDAP messages that flow between the client and the LDAP server

The `ldap_ssl_client_init()` and `ldap_ssl_init()` APIs are provided to initialize the SSL function, and to create a secure SSL connection.

Chapter 2. API categories

Various API categories are supported by IBM Security Directory Server.

The following sets of APIs are supported:
- "LDAP_ABANDON" on page 6
- "LDAP_ADD" on page 7
- "LDAP_BACKUP" on page 9
- "LDAP_BIND / UNBIND" on page 10
- "LDAP_CODEPAGE" on page 18
- "LDAP_COMPARE" on page 25
- "LDAP controls" on page 27
- "LDAP_CREATE_ABORT_TRANSACTION_REQUEST" on page 29
- "LDAP_CREATE_COMMIT_TRANSACTION_REQUEST" on page 30
- "LDAP_CREATE_EFFECTIVE_PWDPOLICY_REQUEST" on page 30
- "LDAP_CREATE_GET_FILE_REQUEST" on page 31
- "LDAP_CREATE_LIMIT_NUM_VALUES_CONTROL" on page 32
- "LDAP_CREATE_LOCATE_ENTRY_REQUEST" on page 33
- "LDAP_CREATE_ONLINE_BACKUP_REQUEST" on page 33
- "LDAP_CREATE_PASSWORD_POLICY_BIND_FINALIZE_REQUEST" on page 34
- "LDAP_CREATE_PASSWORD_POLICY_BIND_INIT_REQUEST" on page 35
- "LDAP_CREATE_PERSISTENTSEARCH_CONTROL" on page 35
- "LDAP_CREATE_PREPARE_TRANSACTION_REQUEST" on page 37
- "LDAP_CREATE_PROXYAUTH_CONTROL" on page 37
- "LDAP_CREATE_RESUME_ROLE_REQUEST" on page 39
- "LDAP_CREATE_RETURN_DELETED_OBJECTS_CONTROL" on page 40
- "LDAP_CREATE_TRANSACTION_CONTROL" on page 40
- "LDAP_CREATE_VLV_CONTROL" on page 41
- "LDAP_DELETE" on page 42
- "LDAP_END_TRANSACTION" on page 43
- "LDAP_ERROR" on page 45
- "LDAP_EXTENDED_OPERATION" on page 50
- "LDAP_FIRST_ATTRIBUTE" on page 52
- "LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE" on page 54
- "LDAP_FREE_LIMIT_NUM_VALUES_RESPONSE" on page 57
- "LDAP_GET_BIND_CONTROLS" on page 57
- "LDAP_GET_DN" on page 58
- "LDAP_GET_TRAN_ID" on page 60
- "LDAP_GET_VALUES" on page 61
- "LDAP_INIT" on page 63
- "LDAP_MEMFREE" on page 79
- "LDAP_MESSAGE" on page 80
- "LDAP_MODIFY" on page 82

- "LDAP_PAGED_RESULTS" on page 85
- "LDAP_PARSE_EFFECTIVE_PWDPOLICY_RESPONSE" on page 88
- "LDAP_PARSE_ENTRYCHANGE_CONTROL" on page 89
- "LDAP_PARSE_EXTENDED_RESULT_W_CONTROLS" on page 91
- "LDAP_PARSE_LIMIT_NUM_VALUES_RESPONSE" on page 91
- "LDAP_PARSE_RESULT" on page 92
- "LDAP_PARSE_VLV_CONTROL" on page 95
- "LDAP_PASSWORD_POLICY" on page 96
- "LDAP_PLUGIN_REGISTRATION" on page 97
- "LDAP_PREPARE_TRANSACTION" on page 100
- "LDAP_RENAME" on page 101
- "LDAP_RESTORE" on page 104
- "LDAP_RESULT" on page 105
- "LDAP_SEARCH" on page 106
- "LDAP_SERVER_INFORMATION IN DNS" on page 112
- "LDAP_SSL" on page 129
- "LDAP_SSL_PKCS11" on page 137
- "LDAP_START_TRANSACTION" on page 139
- "LDAP_START_TLS" on page 140
- "LDAP_STOP_TLS" on page 141
- "LDAP_SSL_ENVIRONMENT_INIT" on page 144
- "LDAP_SORT" on page 145
- "LDAP_SSL_SET_EXTN_SIGALG" on page 150
- "LDAP_SSL_SET_SUITEB_MODE" on page 152
- "LDAP_URL" on page 142

LDAP_ABANDON

Use the `LDAP_ABANDON` API to abandon an LDAP operation in progress.

`ldap_abandon`

`ldap_abandon_ext`

Synopsis

```
#include ldap.h


int ldap_abandon(
       LDAP           *ld,
       int            msgid)

int ldap_abandon_ext(
       LDAP           *ld,
       int            msgid,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

msgid The message ID of an outstanding LDAP operation, as returned by a call to an asynchronous LDAP operation. For example:
- `ldap_search`
- `ldap_modify`, and others.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see “LDAP controls” on page 27.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see “LDAP controls” on page 27.

Usage

The `ldap_abandon()` and `ldap_abandon_ext()` APIs are used to abandon or cancel an LDAP operation in progress. The `msgid` passed must be the message ID of an outstanding LDAP operation, as returned by a call to an asynchronous LDAP operation such as `ldap_search()`, `ldap_modify()`, and others.

Both APIs check to see whether the result of the operation is already returned by the server. If the result of the operation is returned, both APIs delete the result of the operation from the queue of pending messages. If not, both APIs send an LDAP abandon operation to the LDAP server.

The result of an abandoned operation is not returned from a future call to `ldap_result()`.

The `ldap_abandon()` API returns `0` if the abandon was successful or `-1` if unsuccessful; it does not support LDAP V3 server controls or client controls. The `ldap_abandon_ext()` API returns the constant `LDAP_SUCCESS` if the abandon was successful, or another LDAP error code if not.

Errors

`ldap_abandon()` returns `0` if the operation is successful, `-1` if unsuccessful, setting `ld_errno` appropriately. For more information, see “LDAP_ERROR” on page 45. `ldap_abandon_ext()` returns `LDAP_SUCCESS` if successful and returns an LDAP error code if unsuccessful.

See also

ldap_result, ldap_error

LDAP_ADD

Use the `LDAP_ADD` API to carry out an LDAP operation to add an entry.

```
ldap_add
ldap_add_s
ldap_add_ext
ldap_add_ext_s
```

Synopsis

```
#include ldap.h


int ldap_add(
       LDAP          *ld,
       const char    *dn,
       LDAPMod       *attrs[])

int ldap_add_s(
       LDAP          *ld,
       const char    *dn,
       LDAPMod       *attrs[])

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)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

msgid The message ID of an outstanding LDAP operation, as returned by a call to an asynchronous LDAP operation. For example:

- `ldap_search`
- `ldap_modify`, and others.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Usage

The `ldap_abandon()` and `ldap_abandon_ext()` APIs are used to abandon or cancel an LDAP operation in progress. The `msgid` passed must be the message ID of an outstanding LDAP operation, as returned by a call to an asynchronous LDAP operation such as `ldap_search()`, `ldap_modify()`, and others.

Both APIs check to see whether the result of the operation is already returned by the server. If the result of the operation is returned, both APIs delete the result of the operation from the queue of pending messages. If not, both APIs send an LDAP abandon operation to the LDAP server.

The result of an abandoned operation is not returned from a future call to `ldap_result()`.

The ldap_abandon() API returns 0 if the abandon was successful or -1 if unsuccessful; it does not support LDAP V3 server controls or client controls. The ldap_abandon_ext() API returns the constant LDAP_SUCCESS if the abandon was successful, or another LDAP error code if not.

Errors

ldap_abandon() returns 0 if the operation is successful, -1 if unsuccessful, setting ld_errno appropriately. For more information, see “LDAP_ERROR” on page 45. ldap_abandon_ext() returns LDAP_SUCCESS if successful and returns an LDAP error code if unsuccessful.

See also

ldap_result, ldap_error

LDAP_BACKUP

Use the LDAP_BACKUP API to request a server backup. This LDAP routine is used to create and call an LDAP extended operation for requesting a directory server backup.

Synopsis

```
#include ldap.h

    int ldap_backup ( LDAP *ld, Backup_Restore_Result *op_result );
```

Input parameters

ld Specifies the address of the LDAP connection.

op_result
Specifies the address of the result code from the administration server response.

Usage

The ldap_backup routine is a wrapper that is used for creating requests to back up a directory server. This extended operation is only supported by the administration server, ibmdiradm.

If LDAP_SUCCESS is returned for a backup request, the request is sent to the administration server. The administration server submits the backup request unless the directory server is running. It is not configured for online backups or another bulkload, backup, or restore command is already running. The **op_result** parameter indicates the status of the request that is based on the action that is taken by the administration server on the request. The result of backup operation on a directory server does not reflect in the return code or in the **op_result** parameter.

Errors

The ldap_backup routine returns the following error code:

- LDAP_SUCCESS // if the request is submitted
- LDAP_NO_MEMORY // if allocation fails

- LDAP_INSUFFICIENT_ACCESS // DN used for bind does not have authority to send this request
- LDAP_UNWILLING_TO_PERFORM // if backup configuration is not configured for a server
- LDAP_PROTOCOL_ERROR // if request sent to the server is from other than administration server
- LDAP_OTHER // unable to prepare request

See also

ldap_restore

LDAP_BIND / UNBIND

Use the LDAP_BIND / UNBIND API to request a server backup. LDAP routines for binding and unbinding.

ldap_sasl_bind

ldap_sasl_bind_s

ldap_simple_bind

ldap_simple_bind_s

ldap_unbind

ldap_unbind_ext

ldap_unbind_s

ldap_set_rebind_proc

ldap_bind (deprecated)

ldap_bind_s (deprecated)

Synopsis

```
#include ldap.h


int ldap_sasl_bind(
       LDAP           *ld,
       const char     *dn,
       const char     *mechanism,
       const struct berval *cred,
       LDAPControl    **servctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

int ldap_sasl_bind_s(
       LDAP           *ld,
       const char     *dn,
       const char     *mechanism,
       const struct berval *cred,
       LDAPControl    **servctrls,
       LDAPControl    **clientctrls,
       struct berval  **servercredp)

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)

int ldap_unbind(
       LDAP            *ld)

int ldap_unbind_s(
       LDAP            *ld)

int ldap_unbind_ext(
       LDAP            *ld,
       LDAPControl     **servctrls,
       LDAPControl     **clientctrls)

void ldap_set_rebind_proc(
       LDAP            *ld,
       LDAPRebindProc  rebindproc)

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)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

dn Specifies the Distinguished Names (DN) of the entry to bind as.

mechanism
Although various mechanisms are IANA (Internet Assigned Numbers Authority) registered, the only basic mechanisms that are supported by the LDAP library currently are:

- `LDAP_MECHANISM_EXTERNAL` mechanism, represented by the string `EXTERNAL`.
- `LDAP_MECHANISM_GSSAPI` mechanism, represented by the string `GSSAPI`.
- `LDAP_MECHANISM_DIGEST_MD5` mechanism, represented by the string `DIGEST-MD5`.

Note: The CRAM-MD5 mechanism is not supported in a bind operation.

The `LDAP_MECHANISM_EXTERNAL` mechanism indicates to the server that information external to SASL must be used to determine whether the client is authorized to authenticate. For this implementation, the system that provides the external information must be SSL. For example, if the client sets the DN and credentials to NULL (the value of the pointers must be NULL), with mechanism set to `LDAP_MECHANISM_EXTERNAL`, the client is requesting that the server use the authenticated identity from the client X.509 certificate that was used to authenticate the client to the server during the SSL handshake. The server can then use the authenticated identity to access the directory.

The `LDAP_MECHANISM_GSSAPI` mechanism is used to enable Kerberos authentication. In Kerberos authentication, a client presents valid credentials that are obtained from a Kerberos key distribution center (KDC) to an application server. The server decrypts and verifies the credentials using its service key.

When *mechanism* is set to a NULL pointer, the SASL bind request is interpreted as a request for simple authentication, that is, equivalent to using `ldap_simple_bind()` or `ldap_simple_bind_s()`.

For more information about using LDAP client plug-ins, see "LDAP_PLUGIN_REGISTRATION" on page 97. For more information about developing an LDAP client plug-in, see Chapter 5, "LDAP client plug-in programming reference," on page 169.

The `LDAP_MECHANISM_DIGEST_MD5` mechanism is used to authenticate your ID and password with the server by using a challenge or response protocol. The protocol protects the clear-text password over the wire and prevents replay attacks.

This mechanism is useful only when the LDAP server can retrieve the user password. If the password is stored in a hashed form, for example, crypt or SHA, then authentication by using the DIGEST-MD5 mechanism fails. When you use the `DIGEST-MD5` mechanism, the host name that is supplied on the `ldap_init` call must resolve to the fully qualified host name of the server.

The application must supply a user name on the `ldap_sasl_bind_s` call by using the `IBM_CLIENT_MD5_USER_NAME_OID` client control. The application can optionally supply a realm on the `ldap_sasl_bind_s` call by using the `IBM_CLIENT_MD5_REALM_NAME_OID` client control. The application can optionally supply an authorization ID as the **`dn`** parameter.

`cred` Specifies the credentials with which to authenticate. Arbitrary credentials can be passed with this parameter. In most cases, this credential is the user password. When you use a Simple Authentication Security Layer (SASL) bind, the format and content of the credentials depends on the setting of the mechanism parameter.

`method` Selects the authentication method to use. Specify `LDAP_AUTH_SIMPLE` for simple authentication or `LDAP_AUTH_SASL` for SASL bind. The use of the `ldap_bind` and `ldap_bind_s` APIs is deprecated.

`password`
Specifies the password that is used in association with the DN of the entry in which to bind.

`serverctrls`
Specifies a list of LDAP server controls. For more information about server controls, see "LDAP controls" on page 27.

`clientctrls`
Specifies a list of LDAP client controls. For more information about client controls, see "LDAP controls" on page 27.

`rebindproc`
Specifies the entry-point of a routine that is called to obtain bind credentials that are used when a new server is contacted following an LDAP referral.

Output parameters

`msgidp` This result parameter is set to the message ID of the request if the `ldap_sasl_bind()` call succeeds.

servercredp
: This result parameter is set to the credentials returned by the server. If no credentials are returned, it is set to NULL.

Usage

These routines provide various interfaces to the LDAP bind operation. After you use `ldap_init`, `ldap_ssl_init` or `ldap_open` to create an LDAP handle, a bind can be run before other operations are attempted over the connection. Both synchronous and asynchronous versions of each variant of the bind call are provided.

A bind is optional when you communicate with an LDAP server that supports the LDAP V3 protocol. The absence of a bind is interpreted by the LDAP V3 server as a request for unauthenticated access. A bind is required by LDAP servers that support only the LDAP V2 protocol.

The `ldap_simple_bind()` and `ldap_simple_bind_s()` APIs provide simple authentication, by using a user ID or **dn** and a password that is passed in clear-text to the LDAP API.

The `ldap_bind()` and `ldap_bind_s()` provide general authentication routines, where an authentication method can be chosen. In this toolkit, **method** must be set to `LDAP_AUTH_SIMPLE`. Because the use of these two APIs is deprecated, `ldap_simple_bind` and `ldap_simple_bind_s` must be used instead.

The `ldap_sasl_bind` and `ldap_sasl_bind_s` APIs can be used to run general and extensible authentication over LDAP by using the SASL.

All bind routines take **ld** as their first parameter as returned from `ldap_init`, `ldap_ssl_init`, or `ldap_open`.

Simple authentication
: The simplest form of the bind call is `ldap_simple_bind_s()`. It takes the DN to bind and the user password (supplied in password). It returns an LDAP error indication (see `LDAP_ERROR`). The `ldap_simple_bind()` call is asynchronous, taking the same parameters but initiating only the bind operation and returning the message ID of the request it sent. The result of the operation can be obtained with a subsequent call to `ldap_result()`.

General authentication
: The `ldap_bind()` and `ldap_bind_s()` routines are deprecated.

: The deprecated APIs can be used when the authentication method is selected at run time. They both take an extra method parameter when you select the authentication method to use. However, when you use this toolkit, **method** must be set to `LDAP_AUTH_SIMPLE` to select simple authentication. `ldap_bind()` and `ldap_simple_bind()` return the message ID of the initiated request. `ldap_bind_s()` and `ldap_simple_bind_s()` return an LDAP error indication on unsuccessful completion, or `LDAP_SUCCESS` on successful completion.

SASL authentication
: Five categories of SASL authentication are supported:

- SASL authentication by using the `EXTERNAL` mechanism

- SASL authentication by using the `GSSAPI` mechanism (Kerberos is supported and implemented as a plug-in)
- SASL authentication by using the `DIGEST-MD5` mechanism (implemented as a plug-in)
- SASL authentication by using a user-supplied SASL plug-in library
- SASL authentication by using a `SASL` mechanism that is implemented by the application itself

When the input parameter **`mechanism`** is set to a NULL pointer, the SASL bind request is interpreted as a request for simple authentication, that is, equivalent to using `ldap_simple_bind()` or `ldap_simple_bind_s()`.

Also, the SASL authentication mechanism provides a facility for the LDAP server to return server credentials to the client. An application can obtain the server credentials that are returned from the server in the SASL bind result with the `ldap_parse_sasl_bind_result()` API.

EXTERNAL SASL binds

The primary reason for using the EXTERNAL SASL bind mechanism is to use the client authentication mechanism. This mechanism is provided by SSL to strongly authenticate to the directory server by using the client X.509 certificate. For example, the client application can use the following logic:

- `ldap_ssl_client_init` (initialize the SSL library)
- `ldap_ssl_init` (host, port, name), where name references a public or private key pair in the client key database file
- `ldap_sasl_bind_s (ld, dn=NULL, mechanism=LDAP_MECHANISM_EXTERNAL, cred=NULL)`

A server that supports this mechanism, such as the IBM Directory server, can then access the directory. It uses the authenticated client identity as extracted from the client X.509 certificate.

GSSAPI SASL binds

Kerberos authentication is supported in this release. If the input parameters for `ldap_sasl_bind` or `ldap_sasl_bind_s` are `mechanism==GSSAPI` and `cred==NULL`, then it is assumed that the user already authenticated to a Kerberos security server and obtained a ticket-granting-ticket (TGT), either through a desktop log-on process, or by using a program such as `kinit`. The GSSAPI credential handle used to initiate a security context on the LDAP client side is obtained from the current login context. If the input parameters for these two SASL bind functions are `mechanism==GSSAPI` and `cred!=NULL`, the caller of the functions must provide the GSSAPI credential handle for the LDAP client to initiate a security context with an LDAP server. For example, an LDAP server calls an SASL bind function with a credential handle that the server receives from a client as a delegated credential handle.

DIGEST-MD5 SASL binds

The server accepts SASL bind requests by using the DIGEST-MD5 mechanism. There are two types of DIGEST-MD5 bind requests: Initial Authentication bind requests and Subsequent Authentication bind requests. Initial Authentication is required and supported by

IBM Security Directory Server. Subsequent Authentication support is optional, and is not supported by IBM Security Directory Server.

The server responds to a DIGEST-MD5 SASL bind request with a digest-challenge. The challenge contains the values that are required by RFC2831 section 2.1.1, with the following implementation-specific behavior:

- `realm` - The server always sends the realm that the server is configured to be in.
- `nonce` - The server generates a random nonce.
- `qop-options` - The server supports “auth” only.

The next response from the client to the server must be another DIGEST-MD5 SASL bind message. The response includes several fields with values that the server uses as follows:

- `username`- The server uses the user name value to determine whether the user is binding as an administrator or to find an entry in the primary `rdbm`. If the user name is an administrator `DigestUsername`, then the server uses that administrator to bind. If the user name was not an administrator, then the server searches the primary `rdbm` for a user with that user name. If the user name does not correspond to a single entry or the entry does not have a user-password value, the server returns `LDAP_INVALID_CREDENTIALS`. It also prints the appropriate error message.
- `realm` - The value in the realm field must match the realm that the server is configured to be in. If the realm value does not match the realm that the server is configured to be in, the server returns `LDAP_PROTOCOL_ERROR`.
- `nonce` - The value in the nonce field must match the nonce value that the server sent the client with the digest-challenge. If the value does not match, the server returns `LDAP_PROTOCOL_ERROR`.
- `response` - The value in the response field contains a hash of the password. For each of the user-password values that the server gets from the user entry, it generates the DIGEST-MD5 hash. The server then compares it with the hash from the client. If one matches, the server returns `LDAP_SUCCESS` and the user is bound as that user. Otherwise, the server returns `LDAP_INVALID_CREDENTIALS` and print an error message.
- `authzid` - The value in the `authzid` field contains a `"dn:"-` or `"u:"-style` authorization ID from RFC 2829. The server uses RFC 2829 for authority checking after the bind, rather than the entry found for the user name, similar to Proxied Authentication. The entry that the user name corresponds to must have the authority to use the other DN. The server maps the value to an entry similar to the user name parameter if the `authzid` contains a `"u:"-style` authorization ID. If the mapping fails, the server returns `LDAP_INVALID_CREDENTIALS`.

User-supplied SASL plug-ins

The application developer, or a third party, can implement more SASL mechanisms by using the IBM Security Directory Server C-Client SASL plug-in facility. For example, a client and server SASL plug-in can be developed that supports a new authentication mechanism that is based upon a retinal scan. If the mechanism associated with this new authentication mechanism is `retscan`, the application calls `ldap_sasl_bind()` with mechanism set to `retscan`. Depending on how the mechanism and plug-in are designed, the application might be required to also supply the user's DN and credentials. Alternatively, the plug-in itself might be responsible for obtaining the user identity and credentials, which are derived in some way from a retinal scan image.

If the retinal scan plug-in is not defined in `ibmldap.conf`, the application must explicitly register the plug-in, by using the ldap_register_plugin() API. For information about defining a SASL plug-in for use with an application, see the section, *Defining a SASL plug-in* in "LDAP_BIND / UNBIND" on page 10. For more information about using an LDAP client plug-in, see "LDAP_PLUGIN_REGISTRATION" on page 97. For more information about developing an LDAP client plug-in, see Chapter 5, "LDAP client plug-in programming reference," on page 169.

try

SASL mechanisms that are implemented by the application

In some cases, the SASL mechanism might not require the presence of a plug-in, or any special support in the LDAP library. If the application can call the `ldap_sasl_bind()` or `ldap_sasl_bind_s()` API with the parameters appropriate to the mechanism, the LDAP library encodes the SASL bind request and sends it to the server. If a plug-in is defined for the specified mechanism, the request is diverted to the plug-in. The request can do more processing before it sends the SASL bind to the server.

SASL mechanisms that are supported by the LDAP server

The application can query the LDAP server root DSE, by using `ldap_search()` with the following settings:

- base DN set to NULL
- scope that is set to base
- filter that is set to `"(objectclass=*)"`

If the LDAP server supports one or more SASL mechanisms, the search results include one or more values for the *supportedsaslmechanisms* attribute type.

Defining a SASL plug-in

When the application issues an `ldap_sasl_bind_s()` API with a mechanism that is supported by a particular SASL plug-in, the LDAP library must be able to locate the plug-in shared library. Two mechanisms are available for making an LDAP client plug-in that is known to the LDAP library:

- The plug-in for the specified SASL mechanism is defined in the `ibmldap.conf` file.
- The plug-in is explicitly registered by the application, by using the `ldap_register_plugin()` API.

For more information about locating a plug-in library and defining plug-ins in the `ibmldap.conf` file, see the section *Finding the plug-in library* in "LDAP_PLUGIN_REGISTRATION" on page 97.

Unbinding

ldap_unbind_ext(), ldap_unbind(), and ldap_unbind_s() are synchronous APIs. They send an unbind request to the server. Then, they close all open connections that are associated with the LDAP session handle. Later, they dispose of all resources that are associated with the session handle before a return. 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 not valid to make any further LDAP API calls by using the ld.

The ldap_unbind() and ldap_unbind_s() APIs behave identically. The ldap_unbind_ext() API allows server and client controls to be included explicitly. Because there is no server response to an unbind request, therefore you cannot receive a response to a server control sent with an unbind request.

Rebinding while following referrals

The ldap_set_rebind_proc() call is used to set the entry-point of a routine that is called back to obtain bind credentials for use when a new server is contacted following an LDAP referral or search reference. This function is available only when LDAP_OPT_REFERRALS is set, which is the default setting. If ldap_set_rebind_proc() is never called, or if it is called with a NULL **rebindproc** parameter, an unauthenticated simple LDAP bind is always done when you chase referrals. The SSL characteristics of the connections to the referred servers are preserved when you chase referrals. In addition, if the original bind was an LDAP V3 bind, an LDAP V3 bind is used to connect to the referred servers. If the original bind was an LDAP V2 bind, an LDAP V2 bind is used to connect to each referred server.

rebindproc must be a function that is declared as follows:

```
int rebindproc( LDAP *ld, char **whop, char **credp,

    int *methodp, int freeit );
```

The LDAP library first calls the **rebindproc** to obtain the referral bind credentials, and the freeit parameter is zero. The **whop**, **credp**, and **methodp** parameters must be set as appropriate. If the **rebindproc** returns LDAP_SUCCESS, referral processing continues, and the **rebindproc** is called a second time with freeit nonzero to give the application a chance to free any memory that is allocated in the previous call.

If anything other than LDAP_SUCCESS is returned by the first call to the **rebindproc**, referral processing is stopped and the error code is returned for the original LDAP operation.

Errors

Asynchronous routines return -1 in case of error. However, in the case of the asynchronous bind routine ldap_sasl_bind(), it returns LDAP result code other than LDAP_SUCCESS if the sent request was unsuccessful. To obtain the LDAP result code of the asynchronous bind routine, ldap_sasl_bind(), use the ldap_result() API. To obtain the LDAP error, use the ldap_get_errno() API. Synchronous routines return the LDAP error code that results from the operation.

See also

ldap_error, ldap_open

LDAP_CODEPAGE

Use the `LDAP_CODEPAGE` API to manage string conversions. It functions for managing the conversion of strings between UTF-8 and a local code page.

`ldap_xlate_local_to_utf8`

`ldap_xlate_utf8_to_local`

`ldap_xlate_local_to_unicode`

`ldap_xlate_unicode_to_local`

`ldap_set_locale`

`ldap_get_locale`

`ldap_set_iconv_local_codepage`

`ldap_get_iconv_local_codepage`

`ldap_set_iconv_local_charset`

`ldap_char_size`

Synopsis

```
#include ldap.h


int  ldap_xlate_local_to_utf8(
       char          *inbufp,
       unsigned long  *inlenp,
       char          *outbufp,
       unsigned long  *outlenp);

int  ldap_xlate_utf8_to_local(
       char          *inbufp,
       unsigned long  *inlenp,
       char          *outbufp,
       unsigned long  *outlenp);

int  ldap_xlate_local_to_unicode(
       char          *inbufp,
       unsigned long  *inlenp,
       char          *outbufp,
       unsigned long  *outlenp);

int  ldap_xlate_unicode_to_local(
       char          *inbufp,
       unsigned long  *inlenp,
       char          *outbufp,
       unsigned long  *outlenp);
int  ldap_set_locale(
       const char          *locale);

char *ldap_get_locale( )

int  ldap_set_iconv_local_codepage
       char          *codepage);

char *ldap_get_iconv_local_codepage( );

int  ldap_set_iconv_local_charset(
       char          *charset);

int  ldap_char_size(
       char          *p);
```

Input parameters

inbufp
A pointer to the address of the input buffer that contains the data to be translated.

inlenp Length in bytes of the `inbufp` input buffer.

outbufp
A pointer to the address of the output buffer for translated data.

outlenp
Length in bytes of the `outbufp` input buffer.

Note: The output buffer must be three times as large as the input buffer if you want to translate the entire input buffer in a single call.

charset
Specifies the character set to be used when you convert strings between UTF-8 and the local code page. See "IANA character sets supported by platform" on page 192 for the specific `charset` values that are supported for each operating system platform.

Note: The supported values for `charset` are the same values that are supported for the `charset` tag that is optionally defined in Version 1 LDIF files.

codepage
Specifies a code page or code set for overriding the active code page for the currently defined locale. See the system documentation for the code pages that are supported for a particular operating system.

locale Specifies the locale to be used by LDAP when you convert to and from UTF-8 or Unicode. If the locale is not explicitly set, the LDAP library uses the default locale of the application. To force the LDAP library to use another locale, specify the appropriate locale string.

For applications that run on the Windows platform, supported locales are defined in `ldaplocale.h`. For example, the following code is an excerpt from `ldaplocale.h` and shows the available French locales:

```
/*      French - France                                  */
     #define LDAP_LOCALE_FRFR850            "Fr_FR"
     #define LDAP_LOCALE_FRFRISO8859_1      "fr_FR"
```

For applications that run on the AIX operating system, see the locale definitions that are defined in the "Understanding Locale" section of *AIX System Management Guide: Operating System and Devices*. System-defined locales are in `/usr/lib/nls/loc` on the AIX operating system. For example, `Fr_FR` and `fr_FR` are two system-supported French locales.

For Solaris applications, see the system documentation for the set of system-supported locale definitions.

Note: The specified locale is applicable to all conversions by the LDAP library within the applications address space. The LDAP locale is set or changed only when there is no other LDAP activity that occurs within the application on other threads.

p Returns the number of bytes constituting the character pointed to by `p`. For ASCII characters, it is 1. For other character sets, it can be greater than 1.

Output parameters

inbufp
A pointer to the address of the input buffer that contains the data to be translated

inlenp Length in bytes of the `inbufp` input buffer

outbufp
A pointer to the address of the output buffer for translated data

outlenp
Length in bytes of the `outbufp` input buffer

Note: The output buffer must be three times as large as the input buffer if you want to translate the entire input buffer in a single call.

locale When returned from the `ldap_get_locale()` API, locale specifies the currently active locale for LDAP. See the system documentation for the locales that are supported for a particular operating system. For applications that run in the Windows environment, see `ldaplocale.h`.

codepage
When returned from `ldap_get_iconv_local_codepage()` API, code page specifies the currently active code page, as associated with the currently active locale. See the system documentation for the code pages that are supported for a particular operating system.

Usage

These routines are used to manage application-level conversion of data between the local code page and UTF-8. It is used by LDAP when it communicates with an LDAP V3 compliant server. For more information about the UTF-8 standard, see "UTF-8, a Transformation Format of ISO 10646".

When connected to an LDAP V3 server, the LDAP APIs accept and return string data UTF-8 encoded, which is the default mode of operation. Alternatively, your application can rely on the LDAP library to convert LDAP V3 string data to and from UTF-8 by using the ldap_set_option() API to set the `LDAP_OPT_UTF8_IO` option to `LDAP_UTF8_XLATE_ON`. When set, the following connection-based APIs that accept a `ld` as input, expect string data to be supplied as input in the local code page. They return string data to the application in the local code page. In other words, the following LDAP routines and related APIs automatically convert string data to and from the UTF-8 wire protocol:

- `ldap_add` (and family)
- `ldap_bind` (and family)
- `ldap_compare` (and family)
- `ldap_delete` (and family)
- `ldap_parse_reference`
- `ldap_get_dn`
- `ldap_get_values`
- `ldap_modify` (and family)
- `ldap_parse_result`
- `ldap_rename` (and family)
- `ldap_search` (and family)
- `ldap_url_search` (and family)

The following APIs are not associated with a connection, and always expect string data, for example, DNs, to be supplied and returned UTF-8 encoded:

- `ldap_explode_dn`
- `ldap_explode_dns`
- `ldap_explode_rdn`
- `ldap_server_locate`
- `ldap_server_conf_save`
- `ldap_is_ldap_url`
- `ldap_url_parse`
- `ldap_default_dn_set`

The APIs convert your application data to and from the locale code page. There are several reasons for using these APIs:

- The application is using one or more of the non-connection oriented APIs. It requires to convert strings to UTF-8 from the local code page before you use the APIs.
- The application is designed to send and receive strings as UTF-8 when it uses the LDAP APIs. But it requires to convert selected strings to the local code page before you present to the user. When the directory contains heterogeneous data, that is, data is obtained from multiple countries, or locales, it might be the required approach.

These routines are used to manage application-level conversion of data between the local code page and UTF-8. It is used by LDAP when it communicates with an LDAP V3 compliant server. For more information about the UTF-8 standard, see "UTF-8, a Transformation Format of ISO 10646".

When connected to an LDAP V3 server, the LDAP APIs accept and return string data UTF-8 encoded, which is the default mode of operation. Alternatively, your application can rely on the LDAP library to convert LDAP V3 string data to and from UTF-8 by using the `ldap_set_option()` API to set the `LDAP_OPT_UTF8_IO` option to `LDAP_UTF8_XLATE_ON`. When set, the following connection-based APIs that accept a `ld` as input, expect string data to be supplied as input in the local code page. They return string data to the application in the local code page. In other words, the following LDAP routines and related APIs automatically convert string data to and from the UTF-8 wire protocol:

- `ldap_add` (and family)
- `ldap_bind` (and family)
- `ldap_compare` (and family)
- `ldap_delete` (and family)
- `ldap_parse_reference`
- `ldap_get_dn`
- `ldap_get_values`
- `ldap_modify` (and family)
- `ldap_parse_result`
- `ldap_rename` (and family)
- `ldap_search` (and family)
- `ldap_url_search` (and family)

The following APIs are not associated with a connection, and always expect string data, for example, DNs, to be supplied and returned UTF-8 encoded:
- `ldap_explode_dn`
- `ldap_explode_dns`
- `ldap_explode_rdn`
- `ldap_server_locate`
- `ldap_server_conf_save`
- `ldap_is_ldap_url`
- `ldap_url_parse`
- `ldap_default_dn_set`

The APIs convert your application data to and from the locale code page. There are several reasons for using these APIs:
- The application is using one or more of the non-connection oriented APIs. It requires to convert strings to UTF-8 from the local code page before it uses the APIs.
- The application is designed to send and receive strings as UTF-8 when you use the LDAP APIs. But it requires to convert selected strings to the local code page before it presents to the user. When the directory contains heterogeneous data, that is, data is obtained from multiple countries, or locales, it might be the required approach.

If your application might be extracting string data from the directory that originated from other countries or locales, design the application with the following considerations:
- Consider splitting your application into a presentation component, and an LDAP worker component.
  - The presentation component is responsible for obtaining data from external sources. For example, graphical user interfaces (GUIs), command-lines, files, and displaying the data to a GUI, standard out, files. This component typically deals with string data that is represented in the local code page.
  - The LDAP worker component is responsible for interfacing directly with the LDAP programming interfaces. The LDAP worker component can be implemented to deal strictly in UTF-8 when you handle string data. The default mode of operation for the LDAP library is to handle strings that are encoded as UTF-8.
  - String conversion between UTF-8 and the local code page occurs when data is passed to and from the presentation component and the LDAP worker component.

  Consider the following scenario:

  The LDAP worker component issues an LDAP search, and returns a list of entries from the directory. To ensure that no data is lost, the default mode is used and the LDAP library does not convert string data. In this case, it means the DNs of the entries that are returned from the search are represented in UTF-8.

  The application wants to display this list of DNs on a panel. This display can help the user to select the required entry, and the application then retrieves more attributes for the selected DN. Since the DN is represented in UTF-8, it must be converted to the local code page before display.

  The converted DN might not be a faithful representation of the UTF-8 DN. For example, if the DN was created in China, it can contain Chinese characters. If the application is running in a French locale, certain Chinese characters might not be converted correctly, and are replaced with a replacement character.

The application can display the converted DN, but certain characters might be displayed as bobs. Assuming there is enough information for the user to select the wanted DN, the application accesses the LDAP directory with the selected DN for more information. For example, a "jpeg" image so it can display the user photograph. Since "jpeg" images might be large, the application is designed to obtain the jpeg attribute after the user selects the specific DN only.

Ensure that the search gets the "jpeg" attribute by using the selected DN to work. The search must be done with the original UTF-8 version of the selected DN. The search must not be done with the version of the DN that converted to the local code page. This action implies that the application maintains a correlation between the original DN UTF-8 version, and the version that converted to the local code page.

- If the application accepts user input, generate one or more LDAP searches, then display the information without passing the results back into the LDAP library. The application can be designed to provide the LDAP library run the conversions, even though some data loss might theoretically occur. Automatic conversion of string data for a specific `ld` can be enabled by using ldap_set_option() with the `LDAP_OPT_UTF8_IO` option set to `LDAP_UTF8_XLATE_ON`.

`ldap_char_size` returns the number of bytes constituting the character pointed to by p. For ASCII characters, it is 1. For other character sets, it can be greater than 1.

Translate local code page to UTF-8

The `ldap_xlate_local_to_utf8()` API is used to convert a string from the local code page to a UTF-8 encoding. The output string from the conversion process can be larger than the input string. Therefore, the output buffer must be at least twice as large as the input buffer. `LDAP_SUCCESS` is returned if the conversion is successful.

Translate UTF-8 to local code page

The `ldap_xlate_utf8_to_local()` API is used to convert a UTF-8 encoded string to the local code page encoding. The output string from the conversion process can be larger than the input string. Therefore, the output buffer must be at least twice as large as the input buffer. `LDAP_SUCCESS` is returned if the conversion is successful.

Note: Translation of strings from a UTF-8 encoding to local code page can result in loss of data. This loss is possible when one or more characters in the UTF-8 encoding cannot be represented in the local code page. When this translation occurs, a substitution character replaces any UTF-8 characters that cannot be converted to the local code page.

Translate local code page to unicode

The `ldap_xlate_local_to_unicode()` API is used to convert a string from the local code page to the UCS-2 encoding as defined by `ISO/IEC` 10646-1. This same set of characters is also defined in the UNICODE standard. The output string from the conversion process can be larger than the input string. Therefore, the output buffer must be at least twice as large as the input buffer. `LDAP_SUCCESS` is returned if the conversion is successful.

Translate unicode to local code page

The `ldap_xlate_unicode_to_local()` API is used to convert a UCS-2-encoded string to the local code page encoding. The output string from the conversion process can be larger than the input string. Therefore, the output buffer must be at least twice as large as the input buffer. `LDAP_SUCCESS` is returned if the conversion is successful.

Note: Translation of strings from a UCS-2 (UNICODE) encoding to local code page can result in loss of data. This loss is possible when one or more characters in the UCS-2 encoding cannot be represented in the local code page. When this translation occurs, a substitution character replaces any UCS-2 characters that cannot be converted to the local code page.

Set locale

The `ldap_set_locale()` API is used to change the locale that is used by LDAP for conversions between the local code page and UTF-8 (or Unicode). Unless explicitly set with the `ldap_set_locale()` API, LDAP uses the default locale of the application. To force the LDAP library to use another locale, specify the appropriate locale string. For UNIX systems, see the system documentation for the locale definitions. For Windows operating systems, see `ldaplocale.h`.

Get locale

The `ldap_get_locale()` API is used to obtain the active LDAP locale. Values that can be returned are system-specific.

Set code page

The `ldap_set_iconv_local_codepage()` API is used to override the code page that is associated with the active locale. See the system documentation for the code pages that are supported for a particular operating system.

Get code page

The `ldap_get_iconv_local_codepage()` API is used to obtain the code page that is associated with the active locale. See the system documentation for the code pages that are supported for a particular operating system. See “IANA character sets supported by platform” on page 192 for the specific `charset` values that are supported for each operating system platform. The supported values for `charset` are the same values that are supported for the `charset` tag that is optionally defined in Version 1 LDIF files.

Japanese and Korean currency considerations

The generally accepted convention for converting the backslash character (\) (single-byte `X'5C'`) from the Japanese or Korean locale into Unicode is to convert `X'5C'` character to the following considerations:

- the Unicode yen for Japanese
- the Unicode won for Korean

To change the default behavior, set the `LDAP_BACKSLASH` environment variable to `YES` before you use any of the LDAP APIs. When `LDAP_BACKSLASH` is set to `YES`, the X'5C' character is converted to the Unicode (\), instead of the Japanese yen or Korean won.

Errors

Each of the LDAP user configuration APIs returns a nonzero LDAP return code if an error occurs. See “LDAP_ERROR” on page 45 for more details.

See also

ldap_error

LDAP_COMPARE

Use the `LDAP_COMPARE` API to do an LDAP compare operation.

`ldap_compare`

`ldap_compare_s`

`ldap_compare_ext`

`ldap_compare_ext_s`

Synopsis

```
#include ldap.h


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)

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)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

dn Specifies the DN of the entry on which to run the comparison.

attr Specifies the attribute type to use in the comparison.

bvalue
Specifies the attribute value to compare against the entry value. This parameter is used in the `ldap_compare_ext` and `ldap_compare_ext_s` routines, and is a pointer to a struct berval, making it possible to compare binary values. See `LDAP_GET_VALUES`.

serverctrls
: Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
: Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Output parameters

msgidp
: This result parameter is set to the message ID of the request if the `ldap_compare_ext()` call succeeds.

Usage

The various LDAP compare routines are used to run LDAP compare operations. They take `dn`, the DN of the entry upon which to run the compare, and `attr` and `value`, the attribute type, and value to compare to those routines found in the entry.

The `ldap_compare_ext()` API initiates an asynchronous compare operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if it was not successfully sent. If successful, `ldap_compare_ext()` places the message ID of the request in **msgidp*. A subsequent call to `ldap_result()` obtains the result of the operation. After the operation completes, `ldap_result()` returns the status of the operation in the form of an error code. The error code indicates whether the operation completed successfully (`LDAP_COMPARE_TRUE` or `LDAP_COMPARE_FALSE`).

Similarly, the `ldap_compare()` API initiates an asynchronous compare operation and returns the message ID of that operation. Use a subsequent call to `ldap_result()` to obtain the result of the compare. If there is an error, `ldap_compare()` returns `-1`, setting the session error parameters in the LDAP structure appropriately. The session error parameters can be obtained by using `ldap_get_errno()`.

See `LDAP_ERROR` for more details.

Use the synchronous `ldap_compare_s()` and `ldap_compare_ext_s` APIs to run LDAP compare operations. These APIs return an LDAP error code, which can be `LDAP_COMPARE_TRUE` if the entry contains the attribute value and `LDAP_COMPARE_FALSE` if it does not. Otherwise, some error code is returned.

The `ldap_compare_ext()` and `ldap_compare_ext_s()` APIs both support LDAP V3 server controls and client controls.

Errors

`ldap_compare_s()` API returns an LDAP error code that can be interpreted by calling one of the `ldap_error` routines. The `ldap_compare()` API returns `-1` if the initiation request was unsuccessful. It returns the message ID of the request if successful.

See also

ldap_error

LDAP controls

Certain LDAP Version 3 operations can be extended with the use of controls. Controls can be sent to a server or returned to the client with any LDAP message. This type of control is called a server control.

The LDAP API also supports a client-side extension mechanism, which can be used to define client controls. The client-side controls affect the behavior of the LDAP client library and are never sent to the server. The client-side controls are not defined for this client library.

A common data structure is used to represent both server-side and client-side controls:

```
typedef struct ldapcontrol {
        char            *ldctl_oid;
        struct berval   ldctl_value;
        char            ldctl_iscritical;
} LDAPControl, *PLDAPControl;
```

The **LDAPControl** fields have the following definitions:

ldctl_oid
: Specifies the control type, represented as a string.

ldctl_value
: Specifies the data that is associated with the control. The control might not include data.

ldctl_iscritical
: Specifies whether the control is critical or not. If the field is nonzero, the operation is carried out only if it is recognized and supported by the server or the client for client-side controls.

Functions to manipulate controls

The function is to add, remove, or copy controls.

```
ldap_insert_control
ldap_add_control
ldap_remove_control
ldap_copy_controls
```

Synopsis

```
#include ldap.h

int ldap_insert_control(
       LDAPControl *newControl,
       LDAPControl ***ctrlList);
int ldap_add_control(
       const char *oid, ber_len_t len,
       char *value,
       int isCritical,
       LDAPControl ***ctrlList);

int ldap_remove_control(
       LDAPControl *delControl,
       LDAPControl ***ctrlList,
       int freeit);

int ldap_copy_controls(
       LDAPControl ***to_here,
       LDAPControl **from);
```

Input parameters

newcontrol
: Specifies a control to be inserted into a list of controls.

ctrlList
: Specifies a list of LDAP server controls

oid Specifies the control type, represented as a string.

len Specifies the length of the value string.

value Specifies the data that is associated with the control.

isCritical
: Specifies whether the control is critical or not.

delControl
: Specifies the control to be deleted.

freeit Specifies whether to free the control. If set to `TRUE`, the control is freed. If set to `FALSE`, the control is not freed.

to_here
: Specifies the location to which to copy the control list.

from Specifies the location of the control list to be copied.

Usage

The `ldap_insert_control()` API inserts the control **newcontrol* into a list of controls that are specified by ****ctrlList*. The function allocates space in the list for the control, but does not allocate the actual control. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control cannot not be inserted.

The `ldap_add_control()` API creates a control by using the *oid*, *len*, *value* and *isCritical* values, and inserts it into a list of controls that are specified by ****ctrlList*. The function allocates space in the list for the control. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control cannot not be added.

The `ldap_remove_control()` API removes the control from the list. If *freeit* is not 0, the control is freed. If *freeit* is set to 0, the control is not freed. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control cannot not be removed.

The `ldap_copy_controls()` API makes a copy of the control list. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control list cannot not be copied.

LDAP_CREATE_ABORT_TRANSACTION_REQUEST

Use the `LDAP_CREATE_ABORT_TRANSACTION_REQUEST` API or LDAP routine to create a cancel or stop transaction request, in other words, an abort transaction request.

Synopsis

```
#include ldap.h

    struct berval * ldap_create_abort_transaction_request ( const char *tran_id );
```

Input parameters

tran_id
: Specifies the transaction ID as a string.

Output parameters

The `ldap_create_abort_transaction_request()` routine returns a berval struct that contains the abort transaction request.

Usage

This routine is used to create the abort transaction request that can be passed to `ldap_extended_operation()` or `ldap_extended_operation_s()` API.

Errors

If an error is encountered, this routine returns a null value.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control, ldap_create_commit_transaction_request, ldap_create_prepare_transaction_request

LDAP_CREATE_ACCOUNT_STATUS_REQUEST

Use the `LDAP_CREATE_ACCOUNT_STATUS_REQUEST` API or LDAP routine to create a berval for the account status request.

Synopsis

```
#include ldap.h

    struct berval * ldap_create_account_status_request ( char *entryDN );
```

Input parameters

entryDN
: A character string that specifies the entry DN. The **`entryDN`** parameter is set in the berval and must not be null.

Usage

This routine is used by the client to create a berval for an account status extended operation request. This routine creates a berval structure that contains an **`entryDN`**.

Errors

If any errors were encountered, the returned berval is null. If berval request was created successfully, the berval is a valid berval for the account status extended operation.

LDAP_CREATE_COMMIT_TRANSACTION_REQUEST

Use the `LDAP_CREATE_COMMIT_TRANSACTION_REQUEST` API or LDAP routine to create a commit transaction request.

Synopsis

```
#include ldap.h

    struct berval * ldap_create_commit_transaction_request ( const char *tran_id );
```

Input parameters

tran_id
Specifies the transaction ID as a string.

Output parameters

The `ldap_create_commit_transaction_request()` routine returns a berval struct that contains the commit transaction request.

Usage

This routine is used to create a commit transaction request that can be passed to the `ldap_extended_operation()` or `ldap_extended_operation_s()` API.

Errors

If an error is encountered, this routine returns a null value.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control, ldap_create_abort_transaction_request, ldap_create_prepare_transaction_request

LDAP_CREATE_EFFECTIVE_PWDPOLICY_REQUEST

Use the `LDAP_CREATE_EFFECTIVE_PWDPOLICY_REQUEST` API or LDAP routine for creating an extended operation request to query the effective password policy of a user or a group.

Synopsis

```
#include ldap.h

    struct berval *ldap_create_effective_pwdpolicy_request (char *dn);
```

Input parameters

dn Specifies the DN of a user or a group entry.

Usage

This routine encodes the request value for the effective password policy extended operation. The returned value can be used as an input parameter to `ldap_extended_operation_s` or `ldap_extended_operation` function.

Errors

The `ldap_create_effective_pwdpolicy_request` routine returns an LDAP error code if it encounters an error when it encodes the request.

See also

ldap_extended operation, ldap_extended operation_s, ldap_parse_effective_pwdpolicy_response

LDAP_CREATE_GET_FILE_REQUEST

Use the `LDAP_CREATE_GET_FILE_REQUEST` API or LDAP routine to create a berval request that can be sent on the get file extended operation.

Synopsis

```
#include ldap.h

     struct berval* ldap_create_get_file_request (
           int      fileNumber;
           char*    fileName
     );
```

Input parameters

fileNumber
Specifies the file option that is defined in `ldap.h`. The various file options are listed:

- `Other(0)`
- `V3.ibm.at(1)`, `V3.ibm.oc(2)`,
- `V3.user.at(3)`, `V3.user.oc(4)`,
- `V3.config.at(5)`, `V3.config.oc(6)`,
- `V3.system.at(7)`, `V3.system.oc(8)`,
- `V3.modifiedschema(9)`, `V3.ldapsyntaxes(10)`,
- `V3.matchingrules(11)`,
- `key ring file(12)`, `key database file(13)`

fileName
Specifies the file name when the **`fileNumber`** is 0. The value of this parameter must be set to NULL when the **`fileNumber`** is in the range from 1 through 11 and 13. The **`fileName`** parameter must either be provided as a full path to the file or the system must be able to resolve the file name with the set path for the environment. The **`fileName`** parameter can either be a file that is in the configuration file of the server under the `ibm-slapdIncludeSchema` or `ibm-slapdSchemaAdditions` attributes, or a keytab file of a proxy back-end server.

Usage

The `ldap_create_get_file_request` routine is used to create a berval request that can be sent on the get file extended operation.

Errors

This routine does not return any return code. The berval that is returned is NULL, if the routine encounters any errors.

LDAP_CREATE_LIMIT_NUM_VALUES_CONTROL

Use the `LDAP_CREATE_LIMIT_NUM_VALUES_CONTROL` API or LDAP routine to create the `Limit Number of Attribute Values Control`. This control is used to limit the number of values that are returned for the entire entry.

Synopsis

```
#include ldap.h

    int ldap_create_limit_num_values_control(
              LDAP        *ld,
              int         maxTotalValues,
              int         maxValuesPerAttribute,
              int         returnDetails,
              int         isCritical,
              LDAPControl **control);
```

Input parameters

ld Specifies a pointer to the LDAP structure that represents an LDAP connection.

maxTotalValues
An integer that indicates the maximum number of attribute values that can be returned for an entry.

maxValuesPerAttribute
An integer that indicates the maximum number of attribute values that can be returned for an attribute in an entry.

returnDetails
An integer that indicates the type of response wanted. If the value of `returnDetails` is 0, no response controls are returned with the entries. Otherwise, response controls are returned with the entries.

isCritical
An integer that indicates whether the criticality of the control must be critical or not critical. If the value is 0, the criticality of the control is set to not critical. If the value is non-zero, the criticality of the control is set to critical.

control
Specifies the address of a pointer to an `LDAPControl` structure, where the created control is built if the API is successful.

Usage

The `ldap_create_limit_num_values_control` routine is used for creating the Limit Number of Attribute Values Control.

Errors

The `ldap_create_limit_num_values_control` routine returns an LDAP error code if the routine encounters an error.

The errors that are returned by the `ldap_create_limit_num_values_control` routine are listed:
- `LDAP_PARAM_ERROR` // bad input parameter
- `LDAP_NO_MEMORY` // server is out of memory
- `LDAP_SUCCESS` // operation was successful
- `LDAP_ENCODING_ERROR` // an encoding error was encountered

See also

ldap_parse_limit_num_values_response, ldap_free_limit_num_values_response

LDAP_CREATE_LOCATE_ENTRY_REQUEST

Use the `LDAP_CREATE_LOCATE_ENTRY_REQUEST` API or LDAP routine to create a berval request for the locate entry extended operation.

Synopsis

```
#include ldap.h


struct berval *ldap_create_locate_entry_request (const char *entryDN)
```

Input parameters

entryDN
Specifies the entry DN, for which the location details are to be determined.

Usage

This routine is used by the client to create a berval for the locate entry extended operation request.

Errors

If any errors were encountered, the returned berval is null. If berval request is created successfully, the berval is a valid berval for the group evaluation extended operation.

LDAP_CREATE_ONLINE_BACKUP_REQUEST

Use the `LDAP_CREATE_ONLINE_BACKUP_REQUEST` API or LDAP routine to create a berval request that can be sent on the online backup extended operation.

Synopsis

```
#include ldap.h

    struct berval* ldap_create_online_backup_request (char* directoryPath);
```

Input parameters

directoryPath
: Specifies a path to which the target system has write access. This path is used by the DB2® online backup command to store the backup image. The value of the path must not be NULL.

Usage

The `ldap_create_online_backup_request` routine is used to create a berval that can be sent on the online backup extended operation.

Errors

This routine does not return any return code. The berval that is returned is NULL, if the routine encounters any errors.

LDAP_CREATE_PASSWORD_POLICY_BIND_FINALIZE_REQUEST

Use the `LDAP_CREATE_PASSWORD_POLICY_BIND_FINALIZE_REQUEST` API or LDAP routine for creating a password policy bind finalize request and verifying extended operation request.

Synopsis

```
#include ldap.h


struct berval *ldap_create_password_policy_bind_finalize_request (
                const char *bind_dn,
                const int ldap_rc );
```

Input parameters

bind_dn
: The bind DN that is used for running bind password policy checks.

ldap_rc
: The return code for the bind.

Output parameters

berval The berval struct contains the password policy bind finalize and verify bind extended operation request.

Usage

The `ldap_create_password_policy_bind_finalize_request()` API is used to create the prebind password policy request that is used as input parameter to the `ldap_extended_operation` or `ldap_extended_operation_s function`.

Errors

This routine returns a null value if it encounters an error.

LDAP_CREATE_PASSWORD_POLICY_BIND_INIT_REQUEST

Use the LDAP_CREATE_PASSWORD_POLICY_BIND_INIT_REQUEST API or LDAP routine for creating a password policy bind initialize request and verifying the extended operation request.

Synopsis

```
#include ldap.h


struct berval *ldap_create_password_policy_bind_init_ request (
               const char *bind_dn);
```

Input parameters

bind_dn
The bind DN that is used for running password policy checks.

Output parameters

berval The berval struct contains the password policy bind initialize and the verifying extended operation request.

Usage

The ldap_create_password_policy_bind_init_request() API is used to create the prebind password policy request that can be used as input parameter to the ldap_extended_operation or ldap_extended_operation_s function.

Errors

This routine returns a null value if it encounters an error.

LDAP_CREATE_PERSISTENTSEARCH_CONTROL

Use the LDAP_CREATE_PERSISTENTSEARCH_CONTROL API or LDAP routine to create a persistent search control that can be passed to the ldap_search_ext() or ldap_search_ext_s() function to initiate a persistent search.

Synopsis

```
#include ldap.h


#define LDAP_CHANGETYPE_ADD      1
#define LDAP_CHANGETYPE_DELETE   2
#define LDAP_CHANGETYPE_MODIFY   4
#define LDAP_CHANGETYPE_MODDN    8
#define LDAP_CHANGETYPE_ANY      (1|2|4|8)

#define LDAP_CONTROL_PERSISTENTSEARCH   "2.16.840.1.113730.3.4.3"

int ldap_create_peristentsearch_control(
         LDAP       *ld,
         int        changetypes,
         int        changesonly,
         int        return_echg_ctls,
         char       ctl_iscritical,
         LDAPControl **ctrlp);
```

Input parameters

ld
Specifies the LDAP pointer, which acts as an LDAP session handle, returned by previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

changetypes
Specifies a bit field that indicates the client about the type of changes. The value of the field can be `LDAP_CHANGETYPE_ANY` or any logical-OR combination of one or more of the following values:

- `LDAP_CHANGETYPE_ADD`
- `LDAP_CHANGETYPE_DELETE`
- `LDAP_CHANGETYPE_MODIFY`
- `LDAP_CHANGETYPE_MODDN`

The **changetypes** field corresponds to the `changeType` element of the BER-encoded persistent search control value.

changesonly

It is a boolean field that specifies whether the `searchResultEntry` messages for entries that are changed or all the static entries that match the search criteria must be returned to the client.

If the value is non-zero, the entries that are changed are returned. If zero, all the static entries that match search criteria are returned before the server sends change notification. The **changesonly** field corresponds to the `changesOnly` element of the BER-encoded persistent search control value.

return_echg_ctls
It is a boolean field that specifies the behavior of the server about the returning of an **Entry Change Notification** control with each `searchResultEntry` when an entry is changed. If the value of this field is non-zero, the **Entry Change Notification** controls are requested. If zero, the **Entry Change Notification** controls are not requested. The **return_echg_ctls** field corresponds to the `returnECs` element of the BER-encoded persistent search control value.

ctl_iscritical
Sets the `ctl_iscritical` flag within the resulting `LDAPControl` structure. A non-zero value indicates that the persistent search control is critical and a zero value indicates that this control is not critical.

Output parameters

ctrlp
This result parameter is assigned the address of an `LDAPControl` structure that contains the **Persistent Search** control that is created by this routine.

Note: The caller must free the memory that is occupied by the `LDAPControl` structure after its use by calling `ldap_control_free()`.

Usage

This routine is used to create a persistent search control that can be passed to the `ldap_search_ext()` or `ldap_search_ext_s()` function to initiate a persistent search. If the operation is successful, `LDAP_SUCCESS` is returned.

Errors

This routine returns an LDAP error code if the operation is a failure.

See “LDAP_ERROR” on page 45 for a list of the LDAP error codes.

LDAP_CREATE_PREPARE_TRANSACTION_REQUEST

Use the `LDAP_CREATE_PREPARE_TRANSACTION_REQUEST` API or LDAP routine to create a prepare transaction request.

Synopsis

```
#include ldap.h

     struct berval * ldap_create_prepare_transaction_request(
                          const char *tran_id );
```

Input parameters

tran_id
Specifies the transaction ID as a string.

Output parameters

The `ldap_create_prepare_transaction_request()` routine returns a berval struct containing the prepare transaction request.

Usage

This routine is used to create the prepare transaction request that can be passed to `ldap_extended_operation()` or `ldap_extended_operation_s()` API.

Errors

If an error is encountered, this routine returns a null value.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control, ldap_create_abort_transaction_request, ldap_create_commit_transaction_request

LDAP_CREATE_PROXYAUTH_CONTROL

Use the `LDAP_CREATE_PROXYAUTH_CONTROL` API or LDAP routine to create an LDAP control that allows a bind entity to assume a proxy identity.

`ldap_create_proxyauth_control`

`ldap_proxy_dn_prefix`

Synopsis

```
#include ldap.h


int ldap_create_proxyauth_control(
       LDAP           *ld,
       char     *proxyDN,
       int            iscritical,
       LDAPControl    **controlp)


int ldap_proxy_dn_prefix(
       char           **proxyDN,
       char           *parm)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

proxyDN
Specifies the DN of the entry whose identity the client assumes.

iscritical
Specifies whether the persistent search control is critical to the current operation. This parameter must be set to a non-zero value.

controlp
Pointer to a pointer of a structure that is created by this function. This control must be freed by calling `ldap_control_free()` function, when it is done by using the control.

Usage

This API is used to create an LDAP control that contains the proxy authorization identity. The created proxy authorization control is then included in LDAP operations to request an operation from the server.

Using the proxy authorization control mechanism, a client can bind to the LDAP directory by using its own identity. But is granted proxy authorization rights of another user to access the target directory.

When the LDAP server receives an operation with proxy authorization control, the bind DN is validated against the administrative group or the predefined proxy authorization group. This validation is to determine whether the bind DN must be granted the proxy authorization right. In other words, the bound application client must be a member of the administrative group or proxy authorization group to request a proxy authorization operation.

For a specific DN, the `ldap_proxy_dn_prefix` function ensures that the DN has the proxy DN prefix. The DN is passed in by using the **`param`** parameter. The value is returned by using the **`proxyDN`** parameter. If the passed in DN already has the "`dn:`" prefix, the parameter is copied into the return value. A new string is allocated with the "`dn:`" prefix if the passed in DN does not have the "`dn:`" prefix. The return code can be:

- `LDAP_PARAM_ERROR` if the **`param`** is null
- `LDAP_NO_MEMORY` if the function failed to allocate memory
- `LDAP_SUCCESS` if a new **`proxyDN`** was successfully allocated

If `LDAP_SUCCESS` is returned, it is the responsibility of the caller to free the returned **`proxyDN`**.

Errors

`LDAP_PARAM_ERROR` returns if an invalid parameter was passed.

`LDAP_NO_MEMORY` returns if memory cannot be allocated.

`LDAP_ENCODING_ERROR` returns if an error occurred when you encode the control.

`LDAP_UNAVAILABLE_CRITICAL_EXTENSION` returns if server does not support proxy authorization and **`iscritical`** is set to a non-zero value.

See also

ldap controls, ldap_bind, ldap_search, ldap_modify, ldap_delete, ldap_add

LDAP_CREATE_RESUME_ROLE_REQUEST

Use the `LDAP_CREATE_RESUME_ROLE_REQUEST` API to create a request for resuming the role extended operation. The LDAP routine creates a berval that can be sent by using the proxy back-end server resume role extended operation.

Synopsis

```
#include ldap.h


struct berval* ldap_create_resume_role_request (
   int   RequestType,
   char   *PartitionName,
   char   *ServerName)
```

Input parameters

RequestType
: One of the request types that are defined in `ldap.h`. The request type can have one of the following values:
- All (0)
- Partition (1)
- Server (2)
- ServerInAPartition (3)

PartitionName
: Specifies the partition name for the request. If request value is `1` or `3`, **`PartitionName`** must not be NULL. The partition name is one of the following names that are configured in the configuration file:
- `ibm-slapdProxySplitName=`*`Name`*
- `ibm-slapdProxyPartitionIndex=`*`index value`*
- `ibm-slapdProxySplitName=`*`Name`*

ServerName
: Specifies the server URL for the request. If request value is `2` or `3`, **`ServerName`** must not be NULL.

Usage

This API routine creates a berval that is sent by using the proxy back-end server resume role extended operation.

Errors

This routine does not return any return code. If any errors are encountered, the value of the returned berval is set to NULL.

See also

See "LDAP_ERROR" on page 45 for a list of the LDAP error codes.

LDAP_CREATE_RETURN_DELETED_OBJECTS_CONTROL

Use the `LDAP_CREATE_RETURN_DELETED_OBJECTS_CONTROL` API or LDAP routine to create a return deleted objects control.

Synopsis

```
#include ldap.h

int ldap_create_return_deleted_objects_control(
                       LDAP *ld,
                       int control_iscritical,
                       LDAPControl **control);
```

Input parameters

ld
Specifies an LDAP session handle that is returned by a call to `ldap_init()`.

control_iscritical
Specifies whether the control is critical or not. A nonzero value indicates that the return deleted objects control is critical and a zero value if it is not.

control
Specifies the address of a pointer to an `LDAPControl` structure, where the created control is placed.

Usage

This routine creates a return deleted objects control if the return code is `LDAP_SUCCESS`. The caller must free the memory that is occupied by the `LDAPControl` structure after its use.

Errors

- `LDAP_PARAM_ERROR` // returned if an invalid parameter was passed
- `LDAP_NO_MEMORY` // returned if memory cannot be allocated

See also

LDAP controls

LDAP_CREATE_TRANSACTION_CONTROL

Use the `LDAP_CREATE_TRANSACTION_CONTROL` API or LDAP routine to create a transaction control that is sent by using the update operation within a transaction.

Synopsis

```
#include ldap.h


LDAPControl *ldap_create_transaction_control(
          string       tran_id);
```

Input parameters

tran_id
Specifies the transaction ID in a string format.

Output parameters

This routine returns a transaction control with the transaction ID and is set to the value passed in the routine.

Usage

This routine creates a control that is used with update operation within a transaction.

Errors

If an error occurs, this routine returns a NULL value for the control.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control

LDAP_CREATE_VLV_CONTROL

Use the `LDAP_CREATE_VLV_CONTROL` API or LDAP routine to create a virtual list view request control.

Synopsis

```
#include ldap.h

int ldap_create_vlv_control(
   LDAP           *ld,
   LDAPVLVInfo     *vlvinfop,
   LDAPControl     **ctrlp
);
```

Input parameters

ld
Specifies the LDAP session handle that is returned by a call to `ldap_init()`.

vlvinfop
Contains the address of the **`LDAPVLVInfo`** object whose contents are used for constructing the value of control to be created.

ctrlp
Specifies the result parameter that contains the address of the created Virtual list view control. After its use, it must be freed by the caller by using the `ldap_control_free()` function.

The `ldap_create_vlv_control` routine is used for creating a virtual list view request control. The possible return codes from this routine are as follows:

- `LDAP_SUCCESS` // on success
- `LDAP_PARAM_ERROR` // bad input parameter
- `LDAP_NO_MEMORY` // memory allocation failure
- `LDAP_ENCODING_ERROR` // encoding error

See also

ldap_parse_vlv_control

LDAP_DELETE

Use the `LDAP_DELETE` API or LDAP routine for conducting an LDAP operation to delete a leaf entry.

`ldap_delete`

`ldap_delete_s`

`ldap_delete_ext`

`ldap_delete_ext_s`

Synopsis

```
#include ldap.h


int ldap_delete(
      LDAP          **ld,
      const char     *dn)

int ldap_delete_s(
      LDAP           *ld,
      const char     *dn)

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)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

dn
Specifies the DN of the entry to be deleted.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
: Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Output parameters

msgidp
: This result parameter is set to the message ID of the request if the `ldap_delete_ext()` call succeeds.

Usage

Note: 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.

The `ldap_delete_ext()` API initiates an asynchronous delete operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or returns another LDAP error code if the request was not successful. If successful, `ldap_delete_ext()` places the message ID of the request in **msgidp*. `ldap_result()` returns the status of an operation as an error code. The error code indicates whether the operation completed successfully. The `ldap_parse_result()` API checks the error code.

Similarly, the `ldap_delete()` API initiates an asynchronous delete operation and returns the message ID of that operation. A subsequent call to `ldap_result()` can be used to obtain the result of the `ldap_delete()` operation. If there is an error, `ldap_delete()` returns `-1`, setting the session error parameters in the LDAP structure appropriately. These error parameters can be obtained by using `ldap_get_errno()`.

See "LDAP_ERROR" on page 45 for more details.

Use the synchronous `ldap_delete_s()` and `ldap_delete_ext_s()` APIs to run LDAP delete operations. The results of both operations are output parameters. These routines return either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code returns if the operation was not successful.

Both the `ldap_delete_ext()` and `ldap_delete_ext_s()` APIs both support LDAP V3 server controls and client controls.

Errors

`ldap_delete_s()` returns an LDAP error code that can be interpreted by calling an `ldap_error` routine. The `ldap_delete()` API returns `-1` if the request initiation was unsuccessful. It returns the message ID of the request if successful.

See also

ldap_error

LDAP_END_TRANSACTION

Use the `LDAP_END_TRANSACTION` API or LDAP routine to call an end transaction request.

- `ldap_end_transaction`
- `ldap_end_transaction_s`

Synopsis

```
#include ldap.h


int ldap_end_transaction(
       LDAP           *ld,
       string         tran_id,
       int            abort,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

int ldap_end_transaction_s(
       LDAP           *ld,
       string         tran_id,
       int            abort,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

tran_id
Specifies the transaction ID of the end transaction.

abort
Specifies the request type that is sent to the transaction. The request type can be one from the following list:

- 0 – commit transaction
- 1 – abort transaction

serverctrls
Specifies a list of LDAP server controls.

clientctrls
Specifies a list of LDAP client controls.

Output parameters

msgidp
This parameter contains the message ID of the request.

Usage

This API routine is used to initiate an end transaction request against the server.

Errors

This routine returns an LDAP error code if the operation is unsuccessful.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control

LDAP_ERROR

Use the LDAP_ERROR API or LDAP routine to manage or handle protocol errors.

ldap_get_errno

ldap_get_lderrno

ldap_set_lderrno

ldap_perror (deprecated)

ldap_result2error (deprecated)

ldap_err2string

ldap_get_exterror

Synopsis

```
#include ldap.h


int ldap_get_errno(
         LDAP        *ld);

int ldap_get_lderrno (
         LDAP        *ld,
         char        **dn,
         char        **errmsg);

int ldap_set_lderrno (
         LDAP        *ld,
         int         errnum,
         char        *dn,
         char        *errmsg);

void ldap_perror(
        LDAP         *ld,
        const char   *s);

int ldap_result2error(
        LDAP         *ld,
        LDAPMessage  *res,
        int          freeit);
const char *ldap_err2string(
        int          error);

int ldap_get_exterror(
         LDAP        *ld);
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to ldap_init(), ldap_ssl_init() or ldap_open().

dn
Specifies a DN that identifies an existing entry, indicating how much of the name in the request that is recognized by the server. The DN is returned when an LDAP_NO_SUCH_OBJECT error is returned from the server. The matched DN string must be freed by calling ldap_memfree().

errmsg
Specifies the text of the error message, as returned from the server. The error message string must be freed by calling ldap_memfree().

s
Specifies the message prefix, which is prefixed to the string form of the error code held that is stored under the LDAP structure. The string form of the error is the same string that is returned by a call to ldap_err2string().

res Specifies the result, as produced by `ldap_result()` or `ldap_search_s()`, to be converted to the error code with which it is associated.

freeit Specifies whether the result, `res`, must be freed as a result of calling `ldap_result2error()`. If nonzero, the result, `res`, is freed by the call. If zero, `res` is not freed by the call.

errnum
Specifies the LDAP error code, as returned by `ldap_parse_result()` or another LDAP API call.

Usage

These routines provide interpretation of the various error codes that are returned by the LDAP protocol and LDAP library routines.

The `ldap_get_errno()` and `ldap_get_lderrno()` APIs obtain information for the most recent error that occurred for an LDAP operation. When an error occurs at the LDAP server, the server returns the following information back to the client:
- The LDAP result code for the error that occurred.
- A message that contains any additional information about the error from the server.

If the error occurred because an entry specified by a DN cannot be found, the server might also return the DN portion that identifies an existing entry.

Both APIs return the error result code of the server. Use `ldap_get_lderrno()` to obtain the message and matched DN.

The `ldap_set_lderrno()` API sets an error code and other information about an error in the specified LDAP structure. This function can be called to set error information that is retrieved by subsequent `ldap_get_lderrno()` calls.

The `ldap_result2error()` routine takes **`res`**, a result as produced by `ldap_result()` or `ldap_search_s()`, and returns the corresponding error code. Possible error codes follow. See the tables in the following section. If the **`freeit`** parameter is nonzero, it indicates that the **`res`** parameter must be freed by a call to `ldap_msgfree()` after the error code is extracted. The **ld_errno** field in `ld` is set and returned.

The returned value can be passed to `ldap_err2string()`, which returns a pointer to a character string which is a textual description of the LDAP error code. The character string must not be freed when use of the string is complete.

The `ldap_perror()` routine can be called to print an indication of the error on standard error.

The `ldap_get_exterror()` routine returns the current extended error code that is returned by an LDAP server or other library, such as Kerberos or SSL, for the LDAP session. For some error codes, it might be possible to further interpret the error condition. For example, for SSL errors the extended error code might indicate why an SSL handshake failed.

Errors

The possible values for an LDAP error code are shown in the following tables.

Table 1. Return codes and their description

Dec value	Value	Hex value	Brief description	Detailed description
00	`LDAP_SUCCESS`	00	Success	The request was successful.
00	`LDAP_OPERATIONS_ERROR`	01	Operations error	An operations error occurred.
02	`LDAP_PROTOCOL_ERROR`	02	Protocol error	A protocol violation was detected.
03	`LDAP_TIMELIMIT_EXCEEDED`	03	Time limit that exceeded	An LDAP time limit was exceeded.
04	`LDAP_SIZELIMIT_EXCEEDED`	04	Size limit that exceeded	An LDAP size limit was exceeded.
05	`LDAP_COMPARE_FALSE`	05	Compare false	A compare operation returned false.
06	`LDAP_COMPARE_TRUE`	06	Compare true	A compare operation returned true.
07	`LDAP_STRONG_AUTH_NOT_SUPPORTED`	07	Strong authentication that is not supported	The LDAP server does not support strong authentication.
08	`LDAP_STRONG_AUTH_REQUIRED`	08	Strong authentication that is required	Strong authentication is required for the operation.
09	`LDAP_PARTIAL_RESULTS`	09	Partial results and referral received	Partial results that are only returned.
10	`LDAP_REFERRAL`	0A	Referral returned	Referral returned.
11	`LDAP_ADMIN_LIMIT_EXCEEDED`	0B	Administration limit that exceeded	Administration limit that exceeded.
12	`LDAP_UNAVAILABLE_CRITICAL_EXTENSION`	0C	Critical extension that is not supported	Critical extension is not supported.
13	`LDAP_CONFIDENTIALITY_REQUIRED`	0D	Confidentiality is required	Confidentiality is required.
14	`LDAP_SASLBIND_IN_PROGRESS`	0E	SASL bind in progress	An SASL bind is in progress.
16	`LDAP_NO_SUCH_ATTRIBUTE`	10	No such attribute	The attribute type that is specified does not exist in the entry.
17	`LDAP_UNDEFINED_TYPE`	11	Undefined attribute type	The attribute type that is specified is not valid.
18	`LDAP_INAPPROPRIATE_MATCHING`	12	Inappropriate matching	Filter type that is not supported for the specified attribute.
19	`LDAP_CONSTRAINT_VIOLATION`	13	Constraint violation	An attribute value that is specified violates some constraint. For example, a postal address has too many lines, or a line that is too long.
20	`LDAP_TYPE_OR_VALUE_EXISTS`	14	Type or value exists	An attribute type or attribute value that is specified exists in the entry.
21	`LDAP_INVALID_SYNTAX`	15	Invalid syntax	An attribute value that is not valid was specified.
32	`LDAP_NO_SUCH_OBJECT`	20	No such object	The specified object does not exist in the directory.
33	`LDAP_ALIAS_PROBLEM`	21	Alias problem	An alias in the directory points to a nonexistent entry.
34	`LDAP_INVALID_DN_SYNTAX`	22	Invalid DN syntax	A DN that is syntactically not valid was specified.
35	`LDAP_IS_LEAF`	23	Object is a leaf	The object that is specified is a leaf.
36	`LDAP_ALIAS_DEREF_PROBLEM`	24	Alias dereferencing problem	A problem was encountered when you dereferenced an alias.

Table 1. Return codes and their description (continued)

Dec value	Value	Hex value	Brief description	Detailed description
48	`LDAP_INAPPROPRIATE_AUTH`	30	Inappropriate authentication	Inappropriate authentication was specified. For example, `LDAP_AUTH_SIMPLE` was specified and the entry does not have a `userPassword` attribute.
49	`LDAP_INVALID_CREDENTIALS`	31	Invalid credentials	Invalid credentials were presented. For example, the wrong password.
50	`LDAP_INSUFFICIENT_ACCESS`	32	Insufficient access	The user has insufficient access to run the operation.
51	`LDAP_BUSY`	33	DSA is busy	The DSA is busy.
52	`LDAP_UNAVAILABLE`	34	DSA is unavailable	The DSA is unavailable.
53	`LDAP_UNWILLING_TO_PERFORM`	35	DSA cannot run	The DSA cannot run the operation.
54	`LDAP_LOOP_DETECT`	36	Loop detected	A loop was detected.
64	`LDAP_NAMING_VIOLATION`	40	Naming violation	A naming violation occurred.
65	`LDAP_OBJECT_CLASS_VIOLATION`	41	Object class violation	An object class violation occurred. For example, a "required" attribute was missing from the entry.
66	`LDAP_NOT_ALLOWED_ON_NONLEAF`	42	Operation that is not allowed on nonleaf	The operation is not allowed on a nonleaf object.
67	`LDAP_NOT_ALLOWED_ON_RDN`	43	Operation that is not allowed on RDN	The operation is not allowed on an RDN.
68	`LDAP_ALREADY_EXISTS`	44	Exists	The entry exists.
69	`LDAP_NO_OBJECT_CLASS_MODS`	45	Cannot modify object class	Object class modifications are not allowed.
70	`LDAP_RESULTS_TOO_LARGE`	46	Results too large	Results too large.
71	`LDAP_AFFECTS_MULTIPLE_DSAS`	47	Affects multiple DSAs	Affects multiple DSAs.
80	`LDAP_OTHER`	50	Unknown error	An unknown error occurred.
81	`LDAP_SERVER_DOWN`	51	Cannot contact LDAP server	The LDAP library cannot contact the LDAP server.
82	`LDAP_LOCAL_ERROR`	52	Local error	Some local error occurred. This error is usually a failed memory allocation.
83	`LDAP_ENCODING_ERROR`	53	Encoding error	An error was encountered encoding parameters to send to the LDAP server.
84	`LDAP_DECODING_ERROR`	54	Decoding error	An error was encountered decoding a result from the LDAP server.
85	`LDAP_TIMEOUT`	55	Timed out	A time limit was exceeded while you waited for a result.
86	`LDAP_AUTH_UNKNOWN`	56	Unknown authentication method	The authentication method that is specified on a bind operation is not known.
87	`LDAP_FILTER_ERROR`	57	Bad search filter	An invalid filter that is supplied to `ldap_search`. For example, unbalanced parentheses.
88	`LDAP_USER_CANCELLED`	58	User canceled operation	The user canceled the operation.
89	`LDAP_PARAM_ERROR`	59	Bad parameter to an LDAP routine	An LDAP routine that is called with a bad parameter. For example, a NULL ld pointer, and others.
90	`LDAP_NO_MEMORY`	5A	Out of memory	A memory allocation call, such as *malloc*, failed in an LDAP library routine.

Table 1. Return codes and their description (continued)

Dec value	Value	Hex value	Brief description	Detailed description
91	`LDAP_CONNECT_ERROR`	5B	Connection error	Connection error.
92	`LDAP_NOT_SUPPORTED`	5C	Not supported	Not supported.
93	`LDAP_CONTROL_NOT_FOUND`	5D	Control not found	Control not found.
94	`LDAP_NO_RESULTS_RETURNED`	5E	No results that returned	No results that returned.
95	`LDAP_MORE_RESULTS_TO_RETURN`	5F	More results to return	More results to return.
96	`LDAP_URL_ERR_NOTLDAP`	60	URL does not begin with `ldap://`	The URL does not begin with `ldap://`.
97	`LDAP_URL_ERR_NODN`	61	URL has no DN (required)	The URL does not have a DN (required).
98	`LDAP_URL_ERR_BADSCOPE`	62	URL scope string is invalid	The URL scope string is not valid.
99	`LDAP_URL_ERR_MEM`	63	Cannot allocate memory space	Cannot allocate memory space.
100	`LDAP_CLIENT_LOOP`	64	Client loop	Client loop.
101	`LDAP_REFERRAL_LIMIT_EXCEEDED`	65	Referral limit that exceeded	Referral limit that exceeded.
112	`LDAP_SSL_ALREADY_INITIALIZED`	70	`ldap_ssl_client_init` successfully called previously in this process	The `ldap_ssl_client_init` was successfully called previously in this process.
113	`LDAP_SSL_INITIALIZE_FAILED`	71	Initialization call that failed	SSL Initialization call failed.
114	`LDAP_SSL_CLIENT_INIT_NOT_CALLED`	72	Must call `ldap_ssl_client_init` before you attempt to use SSL connection	Must call `ldap_ssl_client_init` before you attempt to use the SSL connection.
115	`LDAP_SSL_PARAM_ERROR`	73	Invalid SSL parameter previously specified	An SSL parameter that was not valid was previously specified.
116	`LDAP_SSL_HANDSHAKE_FAILED`	74	Failed to connect to SSL server	Failed to connect to SSL server.
117	`LDAP_SSL_GET_CIPHER_FAILED`	75	Not used	Deprecated
118	`LDAP_SSL_NOT_AVAILABLE`	76	SSL library cannot be located	Ensure that `GSKit` is installed.
128	`LDAP_NO_EXPLICIT_OWNER`	80	No explicit owner found	No explicit owner was found.
129	`LDAP_NO_LOCK`	81	Cannot obtain lock	Client library was not able to lock a required resource.

In addition, the following DNS-related error codes are defined in the `ldap.h` file:

Table 2. DNS-related return codes

Dec value	Value	Hex value	Detailed description
133	`LDAP_DNS_NO_SERVERS`	85	No LDAP servers found.
134	`LDAP_DNS_TRUNCATED`	86	Warning: truncated DNS results.
135	`LDAP_DNS_INVALID_DATA`	87	Invalid DNS Data.
136	`LDAP_DNS_RESOLVE_ERROR`	88	Cannot resolve system domain or name server.
137	`LDAP_DNS_CONF_FILE_ERROR`	89	DNS Configuration file error.

The following UTF8-related error codes are defined in the `ldap.h` file:

Table 3. UTF8-related return codes

Dec value	Value	Hex value	Detailed description
160	LDAP_XLATE_E2BIG	A0	Output buffer overflow.
161	LDAP_XLATE_EINVAL	A1	Input buffer that is truncated.
162	LDAP_XLATE_EILSEQ	A2	Unusable input character.
163	LDAP_XLATE_NO_ENTRY	A3	No code set point to map to.
176	LDAP_REG_FILE_NOT_FOUND	B0	NT Registry file not found.
177	LDAP_REG_CANNOT_OPEN	B1	NT Registry cannot open.
178	LDAP_REG_ENTRY_NOT_FOUND	B2	NT Registry entry not found.
192	LDAP_CONF_FILE_NOT_OPENED	C0	Plug-in configuration file not opened.
193	LDAP_PLUGIN_NOT_LOADED	C1	Plug-in library that is not loaded.
194	LDAP_PLUGIN_FUNCTION_NOT_RESOLVED	C2	Plug-in function that is not resolved.
195	LDAP_PLUGIN_NOT_INITIALIZED	C3	Plug-in library not initialized.
196	LDAP_PLUGIN_COULD_NOT_BIND	C4	Plug-in function cannot bind.
208	LDAP_SASL_GSS_NO_SEC_CONTEXT	D0	`gss_init_sec_context` failed.

See also

ldap_memfree, ldap_parse routines

LDAP_EXTENDED_OPERATION

Use the `LDAP_EXTENDED_OPERATION` API or LDAP routine to conduct extended operations and parse extended result.

`ldap_extended_operation`

`ldap_extended_operation_s`

Synopsis

```
#include ldap.h


int     ldap_extended_operation(
                LDAP            *ld,
                const char      *reqoid,
                const struct berval *reqdata,
                LDAPControl     **serverctrls,
                LDAPControl     **clientctrls,
                int             *msgidp)

int     ldap_extended_operation_s(
                LDAP            *ld,
                const char      *reqoid,
                const struct berval *reqdata,
                LDAPControl     **serverctrls,
                LDAPControl     **clientctrls,
                char            **retoidp,
                struct berval   **retdatap)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

reqoid Specifies the dotted-object identifier (OID) text string that identifies the extended operation to be run by the server.

reqdata
Specifies the arbitrary data that is required by the extended operation. If NULL, no data is sent to the server.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the `ldap_extended_operation()` call is successfully sent to the server. To check the result of this operation, call the `ldap_result()` and `ldap_parse_result()` APIs. The server can also return an OID and result data. Because the asynchronous `ldap_extended_operation` does not directly return the results, use `ldap_parse_extended_result()` to get the results.

retoidp
This result parameter is set to point to a character string that is set to an allocated, dotted-OID text string that is returned from the server. This string must be disposed of using the `ldap_memfree()` API. If no OID is returned, **`*retoidp`** is set to NULL.

retdatap
This result parameter is set to a pointer to a berval structure pointer that is set to an allocated copy of the data. This data is returned by the server. This struct berval must be disposed of using `ber_bvfree()`. If no data is returned, **`*retdatap`** is set to NULL.

Usage

The `ldap_extended_operation()` function is used to initiate an asynchronous extended operation, which returns `LDAP_SUCCESS` if the extended operation was successfully sent, or an LDAP error code is returned if the operation was not successful. If successful, the `ldap_extended_operation()` API 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 then be passed to `ldap_parse_extended_result()` to obtain the OID and data that is contained in the response.

The `ldap_extended_operation_s()` function is used to initiate a synchronous extended operation, which returns the result of the operation: either `LDAP_SUCCESS` if the operation was successful, or it returns another LDAP error code if it was not successful. The **`retoid`** and **`retdata`** parameters are provided with the OID and data from the response. If no OID or data was returned, these parameters are set to NULL.

If the LDAP server does not support the extended operation, the server rejects the request. IBM Security Directory Server, version 6.0 and later provide a server plug-in interface that can be used to add extended operation support. For more information, see the *IBM Security Directory Server Server Plug-ins Reference*.

To determine whether the requisite extended operation is supported by the server, get the rootDSE of the LDAP server and check for the supportedExtension attribute. If the values for this attribute include the OID of your extended operation, then the server supports the extended operation. If the supportedExtension attribute is not present in the rootDSE, then the server is not configured to support any extended operations.

A list of OIDs for supported extended operations can be found in Appendix F, "Object Identifiers (OIDs) for extended operations and controls," on page 197.

Errors

The lldap_extended_operation() API returns the LDAP error code for the operation.

The ldap_extended_operation() API returns -1 instead of a valid msgid if an error occurs, setting the session error in the LD structure. The session error can be obtained by using ldap_get_errno().

For more information, see "LDAP_ERROR" on page 45.

Notes

These routines allocate storage. Use ldap_memfree to free the returned OID. Use **ber_bvfree** to free the returned struct berval.

See also

ldap_result, ldap_error

LDAP_FIRST_ATTRIBUTE

Use the LDAP_FIRST_ATTRIBUTE API to step through the LDAP entry attributes.

ldap_count_attributes

ldap_first_attribute

ldap_next_attribute

Synopsis

```
#include ldap.h


int ldap_count_attributes(
       LDAP          *ld,
       LDAPMessage    *entry)

char *ldap_first_attribute(
       LDAP          *ld,
       LDAPMessage    *entry,
       BerElement     **berptr)

char *ldap_next_attribute(
       LDAP          *ld,
       LDAPMessage    *entry,
       BerElement     *berptr)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

entry Pointer to the `LDAPMessage` that represents an entry.

Output parameters

berptrs
This parameter is an output parameter that is returned from `ldap_first_attribute()`. This parameter returns a pointer to a `BerElement` that is allocated to track the current position. It is an input and output parameter for subsequent calls to `ldap_next_attribute()`, where it specifies a pointer to a `BerElement` that is allocated by the previous call to `ldap_first_attribute()`. The `BerElement` structure is opaque to the application.

Usage

The `ldap_count_attributes()` routine returns a count of the number of attributes in an LDAP entry. If a NULL entry is returned from `ldap_first_entry()` or `ldap_next_entry()`, and is passed as input to `ldap_count_attributes()`, `-1` is returned.

The `ldap_first_attribute()` and `ldap_next_attribute()` routines are used to step through the attributes in an LDAP entry.

`ldap_first_attribute()` takes an entry as returned by `ldap_first_entry()` or `ldap_next_entry()` and returns a pointer to a buffer that contains the first attribute type in the entry.

The pointer that is returned by `ldap_first_attribute` in `berptr` must be passed to subsequent calls to `ldap_next_attribute` and is used to step through the entry attributes. When there are no attributes that are left to be retrieved, `ldap_next_attribute()` returns NULL and sets the error code to `LDAP_SUCCESS`. If an error occurs, NULL is returned and an error code is set. The memory that is allocated for the `BerElement` buffer must be freed by using `ldap_ber_free()`.

Therefore, when NULL is returned, the `ldap_get_errno()` API must be used to determine whether an error occurs.

If the caller fails to call `ldap_next_attribute()` enough times to exhaust the list of attributes, the caller is responsible for freeing the `BerElement` pointed to by `berptr` when it is no longer needed by calling `ldap_ber_free()`.

The attribute names that are returned by `ldap_first_attribute()` and `ldap_next_attribute()` are suitable for inclusion in a call to `ldap_get_values()`.

`ldap_next_attribute()` returns a string that contains the name of the next type in the entry. This string must be freed by using `ldap_memfree()` when its use is completed.

The attribute names that are returned by `ldap_next_attribute()` are suitable for inclusion in a call to `ldap_get_values()` to retrieve the attribute values.

Errors

If the `ldap_first_attribute()` call results in an error, then NULL is returned, the error code is set.

The `ldap_get_errno()` API can be used to obtain the error code. For a description about possible error codes, see “LDAP_ERROR” on page 45.

Notes

The `ldap_first_attribute()` and `ldap_next_attribute()` routines allocate memory that might be required to be freed by the caller through `ldap_memfree`.

See also

ldap_first_entry, ldap_get_values, ldap_memfree, ldap_error

LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE

Use the `LDAP_FIRST_ENTRY`, `LDAP_FIRST_REFERENCE` API or LDAP routine for result entry and continuation reference parse and counting routines.

APIs with the “_np” suffix are preliminary implementations, and are not documented in the Internet Draft, "C LDAP Application Program Interface".

`ldap_first_entry`

`ldap_next_entry`

`ldap_count_entries`

`ldap_get_entry_controls_np`

`ldap_first_reference`

`ldap_next_reference`

`ldap_count_references`

`ldap_parse_reference_np`

Synopsis

```
#include ldap.h


LDAPMessage *ldap_first_entry(
                LDAP          *ld,
                LDAPMessage   *result)

LDAPMessage *ldap_next_entry(
                LDAP          *ld,
                LDAPMessage   *entry)

int ldap_count_entries(
                LDAP          *ld,
                LDAPMessage   *result)

int ldap_get_entry_controls_np(
                LDAP          *ld,
                LDAPMessage   *entry
                LDAPControl   ***serverctrlsp)

LDAPMessage *ldap_first_reference(
                LDAP          *ld,
                LDAPMessage   *result)
LDAPMessage *ldap_next_reference(
                LDAP          *ld,
                LDAPMessage   *ref)
                LDAPMessage   *result)

int ldap_count_references(
                LDAP          *ld,
                LDAPMessage   *result)

int ldap_parse_reference_np(
                LDAP          *ld,
                LDAPMessage   *ref,
                char          ***referralsp,
                LDAPControl   ***serverctrlsp,
                int           freeit )
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

result Specifies the result that is returned by a call to `ldap_result()` or one of the synchronous search routines, such as `ldap_search_s()`, `ldap_search_st()`, or `ldap_search_ext_s()`.

entry Specifies a pointer to an entry returned on a previous call to `ldap_first_entry()` or `ldap_next_entry()`.

serverctrlsp
Specifies a pointer to a result parameter that is provided with an allocated array of controls that are copied out of the `LDAPMessage` message. The control array must be freed by calling `ldap_controls_free()`.

ref Specifies a pointer to a search continuation reference returned on a previous call to `ldap_first_reference()` or `ldap_next_reference()`.

referralsp
Specifies a pointer to a result parameter that is provided with the contents of the referrals field from the `LDAPMessage` message. The `LDAPMessage` message indicates zero or more alternative LDAP servers where the request must be tried again. The referrals array must be freed by calling `ldap_value_free()`. Supply NULL for this parameter to ignore the referrals field.

freeit Specifies a Boolean value that determines whether the LDAP result chain, as specified by `ref`, is to be freed. Any nonzero value results in the LDAP result chain that is being freed after the requested information is extracted. Alternatively, the `ldap_msgfree()` API can be used to free the LDAP result chain later.

Usage

These routines are used to parse results that are received from `ldap_result()` or the synchronous LDAP search operation routines `ldap_search_s()`, `ldap_search_st()`, and `ldap_search_ext_s()`.

Processing entries

The `ldap_first_entry()` and `ldap_next_entry()` APIs are used to step through and retrieve the list of entries from a search result chain. When an LDAP operation completes and the result is obtained as described, a list of

LDAPMessage structures is returned. This list is denoted as the search result chain. A pointer to the first of these structures is returned by ldap_result() and ldap_search_s().

The ldap_first_entry() routine is used to retrieve the first entry in a chain of search results. It takes the result that is returned by a call to ldap_result(), ldap_search_s(), ldap_search_st(), or ldap_search_ext_s() and returns a pointer to the first entry in the result.

This pointer must be supplied on a subsequent call to ldap_next_entry() to get the next entry, and others, until ldap_next_entry() returns NULL. The ldap_next_entry() API returns NULL when there are no more entries. The entries that are returned from these calls are used in calls to the routines ldap_get_dn(), ldap_first_attribute(), ldap_get_values(), and others.

The ldap_get_entry_controls_np() routine is used to retrieve an array of server controls that are returned in an individual entry in a chain of search results.

Processing continuation references

The ldap_first_reference() and ldap_next_reference() APIs are used to step through and retrieve the list of continuation references from a search result chain. They return NULL when no more continuation references exist in the result that is set to be returned.

The ldap_first_reference() routine is used to retrieve the first continuation reference in a chain of search results. It takes the result as returned by a call to ldap_result(), ldap_search_s(), ldap_search_st(), or ldap_search_ext_s() and returns a pointer to the first continuation reference in the result.

The pointer that is returned from ldap_first_reference() must be supplied on a subsequent call to ldap_next_reference() to get the next continuation reference.

The ldap_parse_reference_np() routine is used to retrieve the list of alternative servers that are returned in an individual continuation reference in a chain of search results. This routine is also used to obtain an array of server controls that are returned in the continuation reference.

Counting entries and references

The ldap_count_entries() API returns the number of entries that are contained in a search result chain. It can also be used to count the number of entries that remain in a chain if called with a message, entry, or continuation reference that is returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), or ldap_next_reference().

The ldap_count_references() API is used to count the number of continuation references returned. It can also be used to count the number of continuation references that remain in a chain.

Errors

If an error occurs in ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), or ldap_next_reference(), NULL is returned, and ldap_get_errno() API can be used to obtain the error code.

If an error occurs in `ldap_count_entries()` or `ldap_count_references()`, `-1` is returned, and `ldap_get_errno()` can be used to obtain the error code. The `ldap_get_entry_controls_np()` and `ldap_parse_reference_np()` APIs return an LDAP error code directly. For example, `LDAP_SUCCESS` if the call was successful, an LDAP error if the call was unsuccessful.

See "LDAP_ERROR" on page 45 for a description of possible error codes.

See also

ldap_result(), ldap_search(), ldap_first_attribute(), ldap_get_values(), ldap_get_dn()

LDAP_FREE_LIMIT_NUM_VALUES_RESPONSE

Use the `LDAP_FREE_LIMIT_NUM_VALUES_RESPONSE` API or LDAP routine for freeing an `LDAPNumValuesResponse` structure.

Synopsis

```
#include ldap.h

 void ldap_free_limit_num_values_response(
     LDAPNumValuesResponse **numValuesResponse);
```

Input parameters

numValuesResponse
: Specifies the address of a pointer to an `LDAPNumValuesResponse` structure to free. The structure is freed and the pointer is set to NULL.

Usage

The `ldap_free_limit_num_values_response` routine is used for freeing an `LDAPNumValuesResponse` structure.

See also

ldap_parse_limit_num_values_response

LDAP_GET_BIND_CONTROLS

Use the `LDAP_GET_BIND_CONTROLS` API or LDAP routine to allow the client by using `ldap_sasl_bind_s` methods to get controls sent by the server.

```
ldap_get_bind_controls
```

Synopsis

```
int ldap_get_bind_controls LDAP_P(
   LDAP *ld,
  LDAPControl ***bind_controls );
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

bind_controls
: Cannot be NULL.

Output parameters

bind_controls has a copy of the bind controls, or NULL if there are no controls.

Usage

After you call ldap_sasl_bind_s, the application calls ldap_get_bind_controls to get a NULL-terminated array of controls that the server returned on the bind. The caller is responsible for freeing the controls by using ldap_controls_free(). If the caller does not call ldap_sasl_bind_s for the supplied **ld**, the client sets **bind_controls** to NULLreturn.

Errors

LDAP_PARAM_ERROR: If bind_controls=NULL, error code if **ld** not valid.

See also

"LDAP controls" on page 27

LDAP_GET_DN

Use the LDAP_GET_DN API or LDAP routine to handle DN and RDN routines.

ldap_dn2ufn

ldap_get_dn

ldap_explode_dn

ldap_explode_dns

ldap_explode_rdn

Synopsis

```
#include ldap.h


char *ldap_dn2ufn(
        const char *dn)

char *ldap_get_dn(
        LDAP       *ld,
        LDAPMessage *entry)

char **ldap_explode_dn(
         const char *dn,
         int        notypes)

char **ldap_explode_dns(
         const char *dn)

char **ldap_explode_rdn(
         const char *rdn,
         int        notypes)
```

Input parameters

ld
: Specifies the LDAP pointer that is returned by a previous call to ldap_init(), ldap_ssl_init(), or ldap_open().

dn
: Specifies the DN to be exploded (as returned from ldap_get_dn()) or converted to a simple form (as returned from ldap_dn2ufn()).

rdn Specifies the RDN to be exploded (as returned from `ldap_explode_dn()`).

entry Specifies the entry whose `dn` is to be retrieved.

notypes
Specifies whether type names are to be returned for each RDN. If nonzero, the type information is stripped. If zero, the type information is retained. For example, setting **`notypes`** to 1 can result in the RDN "cn=Fido" being returned as Fido.

Usage

The `ldap_dn2ufn()` routine takes a DN and converts it into a simple representation by removing the attribute type that is associated with each RDN. For example, the DN `"cn=John Doe, ou=Widget Division, ou=Austin, o=sample"` is returned in simple form as `"John Doe, Widget Division, Austin, sample"`. Space for the simple name is obtained by the LDAP API, and must be freed by a call to `ldap_memfree()`.

The `ldap_get_dn()` routine takes an entry as returned by `ldap_first_entry()` or `ldap_next_entry()` and returns a copy of the DN entry. Space for the DN is obtained by the LDAP API, and must be freed by a call to `ldap_memfree()`.

The `ldap_explode_dn()` routine takes a DN (as returned by `ldap_get_dn()`) and breaks it up into its component parts. Each part is known as a Relative Distinguished Name, or RDN. The `ldap_explode_dn()` API returns a NULL-terminated array of character strings, each component of which contains an RDN from the DN. The **`notypes`** parameter is used to request that only the RDN values, and not their types, be returned. For example, the DN `"cn=Bob,c=US"` returns an array as either `{"cn=Bob","c=US",NULL}` or `{"Bob","US",NULL}` depending on whether **`notypes`** was 0 or 1. The result can be freed by calling `ldap_value_free()`.

The `ldap_explode_dns()` routine takes a DNS-style DN and breaks it up into its component parts. It returns a NULL-terminated array of character strings. For example, the DN "austin.ibm.com" returns `{ "austin", "ibm", "com", NULL }`. The result can be freed by calling `ldap_value_free()`.

The `ldap_explode_rdn()` routine takes an RDN (as returned by `ldap_explode_dn()`) and breaks it up into its component parts. The `ldap_explode_rdn()` API returns a NULL-terminated array of character strings. The **`notypes`** parameter is used to request that only the component values be returned, not their types. For example, the RDN `"ou=Research + cn=Bob"` returns as either `{"ou=Research", "cn=Bob", NULL}` or `{"Research","Bob", NULL}`, depending on whether **`notypes`** was 0 or 1. The result can be freed by calling `ldap_value_free()`.

The client DN processing functions normalize attribute values that contain compound RDNs, escaped hex representations of UTF-8 characters and ber-encoded values. The functions also check that the DN passed in is in a correct format according to RFC 2253. `ldap_explode_rdn` removes back slashes (\) from in front of special characters.

`ldap_dn2ufn`, `ldap_explode_dn` and `ldap_explode_rdn` normalize attribute values by doing the following changes:

- A back slash followed by a two-digit hex representation of a UTF-8 character is converted to the character representation. For example, `cn=\4A\6F\68\6E Doe` is converted to `cn=John Doe`.

- A ber-encoded value is converted to a UTF-8 value. For example, `cn=#04044A6F686E20446F65` is converted to `cn=John Doe`.

`ldap_dn2ufn`, `ldap_explode_dn` and `ldap_explode_rdn` check that the DN passed in is valid. If the DN is not valid, NULL is returned. A DN is not valid if the attribute type or value are in invalid formats. For more information, see RFC 2253.

`ldap_dn2ufn`, `ldap_explode_dn`, and `ldap_explode_rdn` handle compound RDNs. For example:

- The DN `cn=John+sn=Doe` passed into `ldap_dn2ufn` returns `John+Doe`
- `ldap_explode_dn` with **`notype`** returns `John+Doe`
- `ldap_explode_rdn` with **`notype`** returns `[0]=John [1]=Doe`

`ldap_explode_rdn` removes the back slash from in front of special characters. For example, when you call `ldap_explode_rdn(cn=Doe\<Jane+ou=LDAP+o=sample,1)`, `ldap_explode_rdn` returns:

- `[0] = Doe<Jane`
- `[1] = LDAP`
- `[2] =sample`

Errors

If an error occurs in `ldap_dn2ufn()`, `ldap_get_dn()`, `ldap_explode_dn()`, or `ldap_explode_rdn()`, NULL is returned. If `ldap_get_dn()` returns NULL, the `ldap_get_errno()` API can be used to obtain the error code. See "LDAP_ERROR" on page 45 for a description of possible error codes.

Notes

These routines allocate memory that the caller must deallocate.

See also

ldap_first_entry, ldap_error, ldap_value_free

LDAP_GET_TRAN_ID

Use the `LDAP_GET_TRAN_ID` API or LDAP routine to get the transaction ID from the berval struct that is returned by the start transaction.

Synopsis

```
#include ldap.h


char *ldap_get_tran_id(
         struct berval      *tran_id_bv);
```

Input parameters

tran_id_bv
: Specifies the transaction ID in berval format that is returned by the `ldap_start_transaction` routine.

Output parameters

This routine returns a string value of the transaction ID.

Note: The caller must free the allocated memory that is returned by the call after its use.

Usage

This routine retrieves the transaction ID from the result that is returned by the start transaction.

Errors

If an error occurs, this routine returns a NULL value for the transaction ID.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control

LDAP_GET_VALUES

Use the `LDAP_GET_VALUES` API or LDAP routine to fetch or manipulate values that handle routines.

```
ldap_get_values
ldap_get_values_len
ldap_count_values
ldap_count_values_len
ldap_value_free
ldap_value_free_len
```

Synopsis

```
#include ldap.h


  struct berval {
      unsigned long bv_len;
      char *bv_val;
  };


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 **bvals)
```

```
void ldap_value_free(
          char          **vals)

void ldap_value_free_len(
        struct berval   **bvals)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

attr Specifies the attribute whose values are wanted.

entry Specifies an LDAP entry as returned from `ldap_first_entry()` or `ldap_next_entry()`.

vals Specifies a pointer to a NULL-terminated array of attribute values, as returned by `ldap_get_values()`.

bvals Specifies a pointer to a NULL-terminated array of pointers to berval structures, as returned by `ldap_get_values_len()`.

Usage

These routines are used to retrieve and manipulate attribute values from an LDAP entry as returned by `ldap_first_entry()` or `ldap_next_entry()`.

The values of an attribute can be represented in two forms:
- A NULL-terminated array of strings. This representation is appropriate when the attribute contains string data. For example, a title, description or name.
- A NULL-terminated array of berval structures. This representation is appropriate when the attribute contains binary data. For example, a JPEG file.

String values

Use `ldap_get_values()` to obtain attribute values as an array of strings. The `ldap_get_values()` API takes the entry and the attribute **`attr`** whose values are wanted and returns a NULL-terminated array of character strings that represent the attribute values. The **`attr`** can be an attribute type as returned from `ldap_first_attribute()` or `ldap_next_attribute()`, or if the attribute type is known it can be provided.

The number of values in the array of character strings can be counted by calling `ldap_count_values()`. The array of values that is returned can be freed by calling `ldap_value_free()`.

If your application is designed to rely on the LDAP library to convert LDAP V3 string data from UTF-8 to the local code page (enabled on a per-connection basis by using the `ldap_set_option()` API with the `LDAP_OPT_UTF8_IO`), strings returned in the NULL-terminated array of string values can contain multibyte characters, as defined in the local code page. In this case, the application must use string-handling routines that are properly enabled to handle multibyte strings.

If the attribute values are binary in nature, and thus not suitable to be returned as an array of character strings, the `ldap_get_values_len()` routine can be used instead. It takes the same parameters as `ldap_get_values()` but returns a NULL-terminated array of pointers to berval structures, each containing the length of, and a pointer to, a value.

Binary values

The number of values in the array of bervals can be counted by calling `ldap_count_values_len()`. The array of values that is returned can be freed by calling `ldap_value_free_len()`.

Errors

If an error occurs in `ldap_get_values()` or `ldap_get_values_len()`, NULL is returned and the `ldap_get_errno()` API can be used to obtain the error code. See `LDAP_ERROR` for a description of possible error codes.

See also

ldap_first_entry, ldap_first_attribute, ldap_error

LDAP_INIT

Use the `LDAP_INIT` or LDAP routine to initialize the LDAP library and open a connection to an LDAP server, and get or set options for an LDAP connection.

`ldap_init`

`ldap_open` (deprecated)

`ldap_set_option`

`ldap_get_option`

`ldap_version`

Synopsis

```
#include ldap.h


LDAP *ldap_init(
        const char *host,
        int      port)

LDAP *ldap_open(
        const char *host,
        int      port)

int ldap_set_option(
        LDAP     *ld,
        int      optionToSet,
        void     *optionValue)

int ldap_get_option(
        LDAP     *ld,
        int      optionToGet,
        void     *optionValue)

int ldap_version(
        LDAPVersion  *version)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

host Several methods are supported for specifying one or more target LDAP servers, including the following methods:

Explicit Host List
Specifies the name of the host on which the LDAP server is running. The host parameter can contain a blank-separated list of

hosts to try to connect to, and each host can optionally be of the form `host:port`. If present, the `:port` overrides the port parameter that is supplied on `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`. The following examples are typical:

```
ld=ldap_init ("server1", ldap_port);
ld=ldap_init ("server2:1200", ldap_port);
ld=ldap_init ( "server1:800 server2:2000 server3", ldap_port);
```

Localhost
If the host parameter is NULL, the LDAP server is assumed to be running on the local host.

Default Hosts
If the host parameter is set to "`ldap://`" the LDAP library attempts to locate one or more default LDAP servers, with non-SSL ports, by using the `ldap_server_locate()` function. The port that is specified on the call is ignored because `ldap_server_locate()` returns the port. For example, the following settings are equivalent:

```
ld=ldap_init ("ldap://", ldap_port);
```

and

```
ld=ldap_init (LDAP_URL_PREFIX, LDAP_PORT);
```

If more than one default server is located, the list is processed in sequence until an active server is found.

The LDAP URL can include a distinguished name, which is used as a filter for selecting candidate LDAP servers that are based on the server suffixes. If the most significant DN portion is an exact match with a server suffix after normalizing for case, add the server to the candidate servers list. For example, the following example returns default LDAP servers that have a suffix that supports the specified DN only:

```
ld=ldap_init ("ldap:///cn=fred, dc=austin,
       dc=ibm, dc=com", LDAP_PORT);
```

In this case, a server that has a suffix of "`dc=austin, dc=ibm, dc=com`" matches. If more than one default server is located, the list is processed in sequence until an active server is found.

If the LDAP URL contains a host name and optional port, the host is used to create the connection. No attempt is made to locate the default servers, and the DN, if present, is ignored. For example, the following examples are equivalent:

```
ld=ldap_init ("ldap://myserver", LDAP_PORT);
```

and

```
ld=ldap_init ("myserver", LDAP_PORT);
```

For more information about the algorithm that is used to locate default LDAP servers, see "Locating default LDAP servers" on page 78.

Local Socket
If the host parameter is prefixed with a forward slash (/), the host parameter is assumed to be the name of a UNIX socket, that is, family is `AF_UNIX`, and port is ignored. Use of a UNIX socket

requires the LDAP server to be running on the local host. In addition, the local operating system must support UNIX sockets and the LDAP server must be listening on the specified UNIX socket. UNIX variants of the IBM Security Directory Server listen on the `/tmp/s.slapd` local socket, in addition to any configured TCP/IP ports. For example:

```
ld=ldap_init ("/tmp/s.slapd", ldap_port);
```

Host with Privileged Port
On platforms that support the `rresvport` function, typically UNIX platforms, if a specified host is prefixed with `"privport://"`, then the LDAP library uses the `rresvport` function to attempt to obtain one of the reserved ports (512 through 1023), instead of an ephemeral port. The search for a reserved port starts at 1023 and stops at 512. If a reserved port cannot be obtained, the function call fails. For example:

```
ld=ldap_init ("privport://server1", ldap_port);
ld=ldap_init ("privport://server2:1200", ldap_port);
ld=ldap_init ("privport://server1:800 server2:2000
  privport://server3", ldap_port);
```

port Specifies the port number to connect to. If the default IANA-assigned port of 389 is wanted, `LDAP_PORT` must be specified. To use the default SSL port 636 for SSL connections, use `LDAPS_PORT`.

optionToSet
Identifies the option value that is to be set on the `ldap_set_option()` call. For the list of supported options, see the following section, *Usage*.

optionToGet
Identifies the option value that is to be queried on the `ldap_get_option()` call. For the list of supported options, see the following section, *Usage*.

optionValue
Specifies the address of the value to set by using `ldap_set_option()` or the address of the storage in which the queried value is returned by using `ldap_get_option()`.

version
Specifies the address of an `LDAPVersion` structure that contains the following returned values:

sdk_version
SDK version, which is multiplied by 100.

protocol_version
Highest LDAP protocol that is supported, multiplied by 100.

SSL_version
SSL version that is supported, multiplied by 100.

security_level
Level of encryption that is supported in bits. Set to `LDAP_SECURITY_NONE` if SSL not enabled.

ssl_max_cipher
A string that contains the default ordered ciphers set that are supported by this installation. For more information about changing the set of ciphers that is used to negotiate the secure connection with the server, see “LDAP_SET_OPTION syntax for LDAP V2 applications” on page 75.

sdk_vendor
: A pointer to a static string that identifies the supplier of the LDAP library. This string must not be freed by the application.

sdk_build_level
: A pointer to a static string that identifies the build level, including the date when the library was built. This string must not be freed by the application.

Usage

The `ldap_init()` API initializes a session with an LDAP server. The server is not contacted until an operation is run that requires the server. It allows various options to be set after initialization, but before actually contacting the host. It allocates an LDAP structure that is used to identify the connection and maintain per-connection information.

Although still supported, `ldap_open()` is deprecated. The `ldap_open()` API allocates an LDAP structure and opens a connection to the LDAP server. Use `ldap_init()` instead of `ldap_open()`.

The `ldap_init()` and `ldap_open()` APIs return a pointer to an LDAP structure, which must be passed to subsequent calls to `ldap_set_option()`, `ldap_simple_bind()`, `ldap_search()`, and others.

The LDAP structure is opaque to the application. Direct manipulation of the LDAP structure must be avoided. The `ldap_version()` API returns the toolkit version (multiplied by 100). It also sets information in the `LDAPVersion` structure. See the section *Input Parameters*.

Setting and getting session settings

The `ldap_set_option()` API sets options for the specified LDAP connection. The `ldap_get_option()` API queries settings that are associated with the specified LDAP connection.

The following session settings can be set and retrieved by using the `ldap_set_option()` and `ldap_get_option()` APIs:

LDAP_OPT_SIZELIMIT
: Get or set maximum number of entries that can be returned on a search operation.

LDAP_OPT_TIMELIMIT
: Get or set maximum number of seconds to wait for search results.

LDAP_OPT_REFHOPLIMIT
: Get or set maximum number of referrals in a sequence that the client can follow.

LDAP_OPT_DEREF
: Get or set rules for following aliases at the server.

LDAP_OPT_REFERRALS
: Get or set whether referrals must be followed by the client.

LDAP_OPT_DEBUG
: Get or set debug options.

LDAP_OPT_SSL_CIPHER
Get or set SSL ciphers to use.

"LDAP_OPT_SSL_CIPHER_EX" on page 70
Specifies a set of `TLS 1.2` ciphers for the `TLS 1.2` protocol.

"LDAP_OPT_SSL_SECURITY_PROTOCOL" on page 71
Specifies a set of one or more protocols to use for secure communication with an LDAP server.

LDAP_OPT_SSL_TIMEOUT
Get or set SSL timeout for refreshing session keys.

LDAP_OPT_REBIND_FN
Get or set address of the `setrebindproc` application procedure.

LDAP_OPT_PROTOCOL_VERSION
Get or set LDAP protocol version to use (V2 or V3).

LDAP_OPT_SERVER_CONTROLS
Get or set default server controls.

LDAP_OPT_CLIENT_CONTROLS
Get or set default client library controls.

LDAP_OPT_UTF8_IO
Get or set mode for converting string data between the local code page and UTF-8.

LDAP_OPT_HOST_NAME
Get current host name (cannot be set).

LDAP_OPT_ERROR_NUMBER
Get error number (cannot be set).

LDAP_OPT_ERROR_STRING
Get error string (cannot be set).

LDAP_OPT_API_INFO
Get API version information (cannot be set).

LDAP_OPT_EXT_ERROR
Get extended error code.

LDAP_OPT_CONNECT_TIMEOUT
Set connect timeout value for LDAP C clients.

See "LDAP_SET_OPTION syntax for LDAP V2 applications" on page 75 for important information if your LDAP application is based on the LDAP V2 APIs and uses the `ldap_set_option()` or `ldap_get_option()` functions. That is, you are using `ldap_open`, or your application uses `ldap_init()` and `ldap_set_option()` to switch from the default of LDAP V3 to use the LDAP V2 protocol and then uses the `ldap_set_option()` or `ldap_get_option()` calls.

Additional details on specific options for `ldap_set_option()` and `ldap_get_option()` are provided in the following sections.

LDAP_OPT_SIZELIMIT

Use the `LDAP_OPT_SIZELIMIT` setting to specify the maximum number of entries that can be returned on a search operation.

Note: The actual size limit for operations is also bounded by the maximum number of entries that the server is configured to return. Therefore, the actual size limit value is the lesser to that value specified on this option and the value that is configured in the LDAP server.

The default `sizelimit` is unlimited, specified with a value of zero, thus deferring to the `sizelimit` setting of the LDAP server.

For example:

```
sizevalue=50;
ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &sizevalue);
ldap_get_option( ld, LDAP_OPT_SIZELIMIT, &sizevalue);
```

LDAP_OPT_TIMELIMIT

Use the `LDAP_OPT_TIMELIMIT` setting to specify the number of seconds to wait for search results.

Note: The actual time limit for operations is also bounded by the maximum time that the server is configured to allow. Therefore, the actual time limit is lesser of the value that is specified on this option and the value that is configured in the LDAP server.

The default is unlimited, which is specified with a value of zero. For example:

```
timevalue=50;
ldap_set_option( ld, LDAP_OPT_TIMELIMIT, &timevalue);
ldap_get_option( ld, LDAP_OPT_TIMELIMIT, &timevalue);
```

LDAP_OPT_REFHOPLIMIT

Use the `LDAP_OPT_REFHOPLIMIT` setting to specify the maximum number of hops that the client library takes when it chases referrals.

The default is 10. For example:

```
hoplimit=7;
ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, &hoplimit);
ldap_get_option( ld, LDAP_OPT_REFHOPLIMIT, &hoplimit);
```

LDAP_OPT_DEREF

Use the `LDAP_OPT_DEREF` setting to specify alternative rules for following aliases at the server.

The default is `LDAP_DEREF_NEVER`.

Supported values:

`LDAP_DEREF_NEVER`	0
`LDAP_DEREF_SEARCHING`	1
`LDAP_DEREF_FINDING`	2
`LDAP_DEREF_ALWAYS`	3

For example:

```
int deref = LDAP_DEREF_NEVER;
ldap_set_option( ld, LDAP_OPT_DEREF, &deref);
ldap_get_option( ld, LDAP_OPT_DEREF, &deref);
```

LDAP_OPT_REFERRALS

Use the `LDAP_OPT_REFERRALS` setting to specify whether the LDAP library automatically follows referrals that are returned by LDAP servers or not.

The API can be set to one of the constants **`LDAP_OPT_ON`** or **`LDAP_OPT_OFF`**. By default, the LDAP client follows referrals. For example:

```
int value;
ldap_set_option( ld, LDAP_OPT_REFFERALS, (void *)LDAP_OPT_ON);
ldap_get_option( ld, LDAP_OPT_REFFERALS, &value);
```

LDAP_OPT_DEBUG

Use the `LDAP_OPT_DEBUG` setting to specify a bitmap that indicates the level of debug trace for the LDAP library.

Supported values:

```
/* Debug levels */

 LDAP_DEBUG_OFF         0x000
 LDAP_DEBUG_TRACE       0x001
 LDAP_DEBUG_PACKETS     0x002
 LDAP_DEBUG_ARGS        0x004
 LDAP_DEBUG_CONNS       0x008
 LDAP_DEBUG_BER         0x010
 LDAP_DEBUG_FILTER      0x020
 LDAP_DEBUG_CONFIG      0x040
 LDAP_DEBUG_ACL         0x080
 LDAP_DEBUG_STATS       0x100
 LDAP_DEBUG_STATS2      0x200
 LDAP_DEBUG_SHELL       0x400
 LDAP_DEBUG_PARSE       0x800
 LDAP_DEBUG_ANY         0xffff
```

For example:

```
int value;
 int debugvalue= LDAP_DEBUG_TRACE | LDAP_DEBUG_PACKETS;
 ldap_set_option( ld, LDAP_OPT_DEBUG, &debugvalue);
 ldap_get_option( ld, LDAP_OPT_DEBUG, &value );
```

LDAP_OPT_CONNECT_TIMEOUT

Use the `LDAP_OPT_CONNECT_TIMEOUT` setting to specify a timeout value in seconds and microseconds for LDAP C clients.

Client library receives timeout value as Struct `timeval` pointer. For example, specify a 10-second timeout value as follows:

```
struct timeval tv = { 10 , 0 };
ldap_set_option( ld, LDAP_OPT_CONNECT_TIMEOUT, &tv );
```

LDAP_OPT_SSL_CIPHER

Use the `LDAP_OPT_SSL_CIPHER` parameter to specify a set of one or more ciphers to be used when it negotiates the cipher algorithm with the LDAP server.

Choose the first cipher in the list that is common with the list of ciphers that are supported by the server. The default value is “352F05040A090306”. You can use the `ldap_set_option()` API to also set the TLS protocols and ciphers for the TLS protocols. See “`LDAP_SET_OPTION`” on page 76.

Note: If you try to get an SSL cipher and you are not running on an SSL/TLS version of IBM Security Directory Server, an error is returned.

Supported ciphers for SSLv3/TLS 1.0:

```
SLAPD_SSL_RC4_MD5_EX "03"
SLAPD_SSL_RC2_MD5_EX "06"
```

```
SLAPD_SSL_RC4_SHA_US "05"
SLAPD_SSL_RC4_MD5_US "04"
SLAPD_SSL_DES_SHA_US "09"
SLAPD_SSL_3DES_SHA_US "0A"
SLAPD_SSL_AES_128_SHA_US "2F"
SLAPD_SSL_AES_256_SHA_US "35"
```

For example:

```
char *setcipher = "090A";
char *getcipher;
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, setcipher);
ldap_get_option( ld, LDAP_OPT_SSL_CIPHER, &getcipher );
```

Use `ldap_memfree()` to free the memory that is returned by the call to `ldap_get_option()`.

LDAP_OPT_SSL_CIPHER_EX

Use `LDAP_OPT_SSL_CIPHER_EX` to specify a set of `TLS 1.2` ciphers for the `TLS 1.2` protocol. To specify one or more `TLS 1.2` ciphers, separate the ciphers by a comma (`,`).

Supported TLS 1.2 ciphers

The following ciphers are supported by the `TLS 1.2` protocol:

```
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

You can use the `ldap_set_option()` API to also set the TLS protocols and ciphers for the TLS protocols. See “`LDAP_SET_OPTION`” on page 76.

Examples

Example 1

To set one or more ciphers for the `TLS 1.2` protocol, use the following example:

```
char *setcipher = "TLS_RSA_WITH_AES_128_CBC_SHA,
                   TLS_RSA_WITH_AES_256_CBC_SHA";
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER_EX, setcipher);
```

LDAP_OPT_SSL_SECURITY_PROTOCOL

Use LDAP_OPT_SSL_SECURITY_PROTOCOL to specify a set of one or more protocols to use for secure communication with an LDAP server.

Supported protocols

The following macros are defined in the ldap.h header file to represent the protocols:

```
#define LDAP_SECURITY_PROTOCOL_ALL "SSLV3,TLS10,TLS11,TLS12"
#define LDAP_SECURITY_PROTOCOL_DEFAULT "SSLV3,TLS10"
#define LDAP_SECURITY_PROTOCOL_SSLV3 "SSLV3"
#define LDAP_SECURITY_PROTOCOL_TLSV10 "TLS10"
#define LDAP_SECURITY_PROTOCOL_TLSV11 "TLS11"
#define LDAP_SECURITY_PROTOCOL_TLSV12 "TLS12"
```

You can use the ldap_set_option() API to also set the TLS protocols and ciphers for the TLS protocols. See "LDAP_SET_OPTION" on page 76.

Examples

Example 1:

To set all protocols for secure communication with an LDAP server, use the following example:

```
ldap_set_option(
          ld, LDAP_OPT_SSL_SECURITY_PROTOCOL,
          LDAP_SECURITY_PROTOCOL_ALL);
```

LDAP_OPT_SSL_TIMEOUT

Use the LDAP_OPT_SSL_TIMEOUT setting to specify in seconds the SSL inactivity timer.

After the number of seconds specified, in which no SSL activity occurs, the SSL connection is refreshed with new session keys. A smaller value can help increase security, but has a small affect on performance. The default SSL timeout value is 43200 seconds. For example:

```
value = 100;
ldap_set_option( ld, LDAP_OPT_SSL_TIMEOUT, &value );
ldap_get_option( ld, LDAP_OPT_SSL_TIMEOUT, &value)
```

Note: If you use **LDAP_OPT_SSL_TIMEOUT** and you are not running on an SSL version of IBM Security Directory Server, an error is returned.

LDAP_OPT_REBIND_FN

Use the LDAP_OPT_REBIND_FN setting to specify the routine address to be called by the LDAP library to authenticate a connection with another LDAP server when it chases a referral or search reference.

If a routine is not defined, referrals are chased by using the identity and credentials that are specified on the bind sent to the original server. A default routine is not defined. For example:

```
extern LDAPRebindProc proc_address;
LDAPRebindProc value;
ldap_set_option( ld, LDAP_OPT_REBIND_FN, &proc_address);
ldap_get_option( ld, LDAP_OPT_REBIND_FN, &value);
```

LDAP_OPT_PROTOCOL_VERSION

Use the LDAP_OPT_PROTOCOL_VERSION setting to specify the LDAP protocol to be used by the LDAP client library when it connects to an LDAP server.

The API is also used to determine which LDAP protocol is being used for the connection. For an application that uses `ldap_init()` to create the LDAP connection, the default value of this option is `LDAP_VERSION3` for communicating with the LDAP server. The default value of this option is `LDAP_VERSION2` if the application uses the deprecated `ldap_open()` API. In either case, the `LDAP_OPT_PROTOCOL_VERSION` option can be used with `ldap_set_option()` to change the default. The LDAP protocol version must be reset before you issue the bind, or any operation that causes an implicit bind. For example:

```
   version2 = LDAP_VERSION2;
   version3 = LDAP_VERSION3;
/* Example for Version 3 application setting version to version 2 */
   ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version2);
/* Example of Version 2 application setting version to version 3 */
   ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version3);
   ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, &value);
```

LDAP_OPT_SERVER_CONTROLS

Use the `LDAP_OPT_SERVER_CONTROLS` setting to specify a default list of server controls to be sent with each request.

The default list can be overridden by specifying a server control, or list of server controls, on specific APIs. By default, there are no settings for server controls. For example:

```
ldap_set_option( ld, LDAP_OPT_SERVER_CONTROLS, &ctrlp);
```

LDAP_OPT_CLIENT_CONTROLS

Use the `LDAP_OPT_CLIENT_CONTROLS` setting to specify a default list of client controls to be processed by the client library with each request.

Because client controls are not defined for this version of the library, the `ldap_set_option()` API can be used to define a set of default, non-critical client controls. If one or more client controls in the set are critical, the entire list is rejected with a return code as follows:

```
LDAP_UNAVAILABLE_CRITICAL_EXTENSION
```

LDAP_OPT_UTF8_IO

Use the `LDAP_OPT_UTF8_IO` setting to specify whether the LDAP library automatically converts string data to and from the local code page.

The API can be set to either `LDAP_UTF8_XLATE_ON` or `LDAP_UTF8_XLATE_OFF`. By default, the LDAP library does not convert string data.

When conversion is disabled by default, the LDAP library assumes that data received from the application by using LDAP APIs is already represented in UTF-8. Similarly, the LDAP library assumes that the application is prepared to receive string data from the LDAP library represented in UTF-8, or as binary.

When `LDAP_UTF8_XLATE_ON` is set, the LDAP library assumes that string data received from the application by using LDAP APIs is in the default or explicitly designated code page. Similarly, all string data that is returned from the LDAP library back to the application is converted to the designated local code page.

Only string data that is supplied on connection-based APIs is translated, that is, only those APIs that include a `ld` are subject to translation.

Translation of strings from a UTF-8 encoding to local code page can result in loss of data. The loss occurs when one or more characters in the UTF-8 encoding cannot be represented in the local code page. When this translation occurs, a substitution character replaces any UTF-8 characters that cannot be converted to the local code page.

For more information about explicitly setting the locale for conversions, see `ldap_set_locale()`. For example:

```
  int value;
  ldap_set_option( ld, LDAP_OPT_UTF8_IO, (void*)LDAP_UTF8_XLATE_ON);
  ldap_get_option( ld, LDAP_OPT_UTF8_IO, &value);
```

LDAP_OPT_HOST_NAME

Use the `LDAP_OPT_HOST_NAME` setting to return a pointer to the host name for the original connection.

This setting is a read-only option that returns a pointer to the host name for the original connection, as specified on `ldap_init()`, `ldap_open()`, or `ldap_ssl_init()`. For example:

```
  char *hostname;
  ldap_get_option( ld, LDAP_OPT_HOST_NAME, &hostname);
```

Use `ldap_memfree` to free the memory that is returned by the call to `ldap_get_option()`.

LDAP_OPT_ERROR_NUMBER

Use the `LDAP_OPT_ERROR_NUMBER` setting to return an error code that is associated with the most recent LDAP error.

This setting is a read-only option. It returns the error code that is associated with the most recent LDAP error that occurred for the specified LDAP connection. For example:

```
int error;
ldap_get_option( ld, LDAP_OPT_ERROR_NUMBER, &error);
```

LDAP_OPT_ERROR_STRING

Use the `LDAP_OPT_ERROR_STRING` setting to return a text message that is associated with the most recent LDAP error.

This setting is a read-only option. It returns the text message that is associated with the most recent LDAP error that occurred for the specified LDAP connection. For example:

```
char *error_string;
ldap_get_option( ld, LDAP_OPT_ERROR_STRING, &error_string);
```

Use `ldap_memfree()` to free the memory that is returned by the call to `ldap_get_option()`.

LDAP_OPT_API_INFO

Use the `LDAP_OPT_API_INFO` setting to return basic information about the API and the specific implementation that is used.

This setting is a read-only option. The **`ld`** parameter to `ldap_get_option()` can be either NULL or a valid LDAP session handle that is obtained by calling `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`. The **`optdata`** parameter to `ldap_get_option()` must be the address of an `LDAPAPIInfo` structure, which is defined as follows:

```
typedef struct ldapapiinfo {
    int  ldapai_info_version;       /* version of this struct (1) */
    int  ldapai_api_version;        /* revision of API supported */
    int  ldapai_protocol_version;   /* highest LDAP version supported */
    char **ldapai_extensions;       /* names of API extensions */
    const char *ldapai_vendor_name; /* name of supplier */
    int  ldapai_vendor_version;     /* supplier-specific version times 100 */
} LDAPAPIInfo;
```

Note: The **ldapai_info_version** field of the `LDAPAPIInfo` structure must be set to the value `LDAP_API_INFO_VERSION` before you call `ldap_get_option()` so that it can be checked for consistency. All other fields are set by the `ldap_get_option()` function.

The members of the `LDAPAPIInfo` structure are:

ldapai_info_version
A number that identifies the version of the `LDAPAPIInfo` structure. This member must be set to the value `LDAP_API_INFO_VERSION` before you call `ldap_get_option()`. If the value received is not recognized by the API implementation, the `ldap_get_option()` function sets `ldapai_info_version` to a valid value that can be recognized, sets `ldapai_api_version` to the correct value, and returns an error without supplying any of the other fields in the `LDAPAPIInfo` structure.

ldapai_api_version
A number that matches that assigned to the `C LDAP API RFC` supported by the API implementation. This number must match the value of the `LDAP_API_VERSION` define.

ldapai_protocol_version
The highest LDAP protocol version that is supported by the implementation. For example, if LDAP V3 is the highest version supported then this field is set to 3.

ldapai_extensions
A NULL-terminated array of character strings that lists the names of API extensions. The caller is responsible for disposing of the memory that is occupied by this array by passing it to `ldap_value_free()`.

LDAP_OPT_EXT_ERROR

Use the `LDAP_OPT_EXT_ERROR` setting to return the extended error code.

This setting is a read-only option. For example, if an SSL error occurred when it attempts to call an `ldap_search_s` API, the actual SSL error can be obtained by using `LDAP_OPT_EXT_ERROR`:

```
int error;
ldap_get_option( ld, LDAP_OPT_EXT_ERROR, &exterror);
```

`LDAP_OPT_EXT_ERROR` returns errors reported by the SSL library.

Errors

You can interpret errors that are returned by the LDAP API routines.

If an error occurs, a nonzero return code is returned from `ldap_set_option` and `ldap_get_option`.

LDAP_DEBUG

Use the `LDAP_DEBUG` API for debugging information from a client application.

To obtain debug information from a client application that is built by using the IBM Security Directory Server LDAP C-API, you can set the environment variables `LDAP_DEBUG` and `LDAP_DEBUG_FILE`.

For UNIX, enter the following command before you run your application:

```
export LDAP_DEBUG=65535
```

For the Windows NT and Windows 2000 operating systems, enter the following command before you run your application:

```
set LDAP_DEBUG=65535
```

Trace messages in the LDAP C-API library are output to standard error. Use `LDAP_DEBUG_FILE=`*`xxxxx`* to send the trace output to the file *xxxxx*.

These environment variables affect only applications that are run in the same shell or command window session. You can also call `ldap_set_option()` in your application to enable and disable the library trace messages.

LDAP_SET_OPTION syntax for LDAP V2 applications

Use the `LDAP_SET_OPTION` API to maintain compatibility with the LDAP client library older versions.

To maintain compatibility with older versions of the LDAP client library (pre-LDAP V3), the `ldap_set_option()` API expects the value of the following option values to be supplied, instead of the address of the value, when the application is running as an LDAP V2 application:

- `LDAP_OPT_SIZELIMIT`
- `LDAP_OPT_TIMELIMIT`
- `LDAP_OPT_SSL_TIMEOUT`
- `LDAP_OPT_DEREF`
- `LDAP_OPT_DEBUG`

You can use the `ldap_set_option()` API to also set the TLS protocols and ciphers for the TLS protocols. See "`LDAP_SET_OPTION`" on page 76.

The value that is returned by `ldap_get_option()` when `LDAP_OPT_PROTOCOL_VERSION` is specified can be used to determine how parameters must be passed to the `ldap_set_option()` call. The easiest way to work with this compatibility feature is to ensure that calls to `ldap_set_option()` are all run while the `LDAP_OPT_PROTOCOL_VERSION` is set to the same value. If it cannot be ensured by the application, then follow the format of the following example when you code the call to `ldap_set_option()`:

```
int sizeLimit=100;

int protocolVersion;

ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, &protocolVersion );

if ( protocolVersion == LDAP_VERSION2 ) {
   ldap_set_option( ld, LDAP_OPT_SIZELIMIT, (void *)sizeLimit );
} else { /* the protocol version is LDAP_VERSION3 */
   ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &sizeLimit );
}
```

An LDAP application is typically running as LDAP V2 when it uses `ldap_open()` to create the LDAP connection. An LDAP application is typically running as LDAP V3 when it uses `ldap_init()` to create the LDAP connection. However, it was possible with the LDAP V2 API to call `ldap_init()`, so there can be cases in which this condition is not true. `LDAP_OPT_PROTOCOL_VERSION` can be used to toggle the protocol, in which case the behavior of `ldap_set_option()` changes.

LDAP_SET_OPTION

Use the ldap_set_option API in an LDAP application to access and set various LDAP session parameters.

Purpose

The `ldap_set_option()` API in an LDAP application sets the parameters for a secure connection with an LDAP server. The `ldap_set_option()` API calls the `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()` API to initialize a session with an LDAP server. After the successful initialization, the `ldap_set_option()` API obtains a pointer to an LDAP structure. You must call this API before you issue a bind or any other operations that connects to the server.

Synopsis

```
#include <ldap.h>

int ldap_set_option(
       LDAP *ld,
       int optionToSet,
       void *optionValue)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

optionToSet
Identifies the value for the `ldap_set_option()` call.

optionValue
Specifies the address of the value to set with `ldap_set_option()`.

Session settings

In versions earlier than IBM Security Directory Server, version 6.3, Fix Pack 17, you can set ciphers for `SSLv3/TLS 1.0` protocol suite with `ldap_set_option()`. For IBM Security Directory Server, version 6.3, Fix Pack 17 or later, you can set the `SSLv3`, `TLS 1.0`, `TLS 1.1`, or `TLS 1.2` protocols and the `TLS 1.2` ciphers with `ldap_set_option()`. You can set the options for an LDAP connection with `ldap_set_option()`.

You can set the following session settings with `ldap_set_option()`:

LDAP_OPT_SSL_CIPHER
Specifies a set of ciphers for the `SSLv3`, `TLS 1.0`, or `TLS 1.1` protocols.

Table 4. Supported ciphers for the SSLv3, TLS 1.0, or TLS 1.1 protocol

Ciphers	Supported by SSLv3 and TLS 1.0	Supported by TLS 1.1
SLAPD_SSL_RC4_MD5_EX "03"	Yes	No
SLAPD_SSL_RC2_MD5_EX "06"	Yes	No
SLAPD_SSL_RC4_SHA_US "05"	Yes	Yes
SLAPD_SSL_RC4_MD5_US "04"	Yes	Yes
SLAPD_SSL_DES_SHA_US "09"	Yes	Yes
SLAPD_SSL_3DES_SHA_US "0A"	Yes	Yes
SLAPD_SSL_AES_128_SHA_US "2F"	Yes	Yes
SLAPD_SSL_AES_256_SHA_US "35"	Yes	Yes

Note: The ciphers with 03 and 06 hexadecimal values are not supported by the TLS 1.1 protocol.

To set one or more ciphers for the SSLv3, TLS 1.0, or TLS 1.1 protocol, use the following example:

```
char *setcipher = "352F090A";
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, setcipher);
```

LDAP_OPT_SSL_CIPHER_EX

Specifies a set of TLS 1.2 ciphers for the TLS 1.2 protocol. To specify one or more TLS 1.2 ciphers, separate the ciphers by a comma (,).

The following ciphers are supported by the TLS 1.2 protocol:

```
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
```

To set one or more ciphers for the TLS 1.2 protocol, use the following example:

```
char *setcipher = "TLS_RSA_WITH_AES_128_CBC_SHA,
                   TLS_RSA_WITH_AES_256_CBC_SHA";
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER_EX, setcipher);
```

LDAP_OPT_SSL_SECURITY_PROTOCOL

Specifies a set of one or more protocols to use for secure communication with an LDAP server.

The following macros are defined in the `ldap.h` header file to represent the protocols:

```
#define LDAP_SECURITY_PROTOCOL_ALL "SSLV3,TLS10,TLS11,TLS12"
#define LDAP_SECURITY_PROTOCOL_DEFAULT "SSLV3,TLS10"
#define LDAP_SECURITY_PROTOCOL_SSLV3 "SSLV3"
#define LDAP_SECURITY_PROTOCOL_TLSV10 "TLS10"
#define LDAP_SECURITY_PROTOCOL_TLSV11 "TLS11"
#define LDAP_SECURITY_PROTOCOL_TLSV12 "TLS12"
```

To set all protocols for secure communication with an LDAP server, use the following example:

```
ldap_set_option(
          ld, LDAP_OPT_SSL_SECURITY_PROTOCOL,
          LDAP_SECURITY_PROTOCOL_ALL);
```

Errors

The `ldap_set_option()` AP returns a non-zero value if an error occurs. To obtain a detailed error report, you must run the application in debug mode and check the debug traces.

Locating default LDAP servers

You can search for default LDAP servers with the LDAP APIs.

When the `ldap_init()`, `ldap_open()`, or `ldap_ssl_init()` APIs are called with an LDAP URL of the following forms, the `ldap_server_locate()` function is used to obtain a set of one or more default LDAP servers:

```
ld=ldap_init ("ldap://", ldap_port);          /* locate servers with
       non-secure ports */
ld=ldap_ssl_init ("ldaps://", ldap_port);     /* locate servers with
       secure SSL ports */
```

The `ldap_server_locate()` API provides several options for searching for default LDAP servers. An application by using `ldap_server_locate()` in an explicit fashion can control these options. When `ldap_server_locate()` is used implicitly, as described here, the following options are used:

Security
: If the non-secure LDAP URL is specified such as `ldap://`, servers with a non-secure security type are used as candidate servers only. If the secure LDAP URL is specified such as `ldaps://`, servers with a secure security type are used as candidate servers only.

Source for Server Information
: The `ldap_server_locate()` API can be used to find default LDAP server information in either a local configuration file, or published in the Domain Name System (DNS). In this case, the default behavior is used. The `ldap_server_locate()` API looks for a local configuration file first, and attempts to find one or more LDAP servers that meet the search criteria (security and suffix filter). If nothing is found, it then searches DNS. For more information about using a local configuration file, see `ldap_server_conf_save()`.

DNS Domain Name
: When you search the local configuration and DNS, the `ldap_server_locate()` API assumes that your default LDAP servers are published in your locally configured TCP or DNS. For example, ibm.com.

Service Name and Protocol
A complete search is run by using `ldap` for the service name and `tcp` for the protocol. If no servers are located, the search is rerun by using `_ldap` and `_tcp`.

Note: If the default behavior as described here is not appropriate for your application, consider by using the `ldap_server_locate()` API explicitly before you call the `ldap_init()` or `ldap_ssl_init()` API.

Multithreaded applications

A multi-threaded application that uses system or library calls can use the `ldap_get_errno()` to indicate the error type.

The LDAP client library is re-entrant. While a multithreaded application can safely use the LDAP library on multiple threads within the application, there are a few considerations to keep in mind:

- The `ldap_get_errno()` API obtains information about the most recent error that occurred on the current thread for the specified LDAP connection. It does not return the most recent LDAP error that occurred on any thread.
- If an operation results in more than one response message from a server, then all the response messages are returned to only one thread. The thread that reads the first response message for that operation must read all the remaining response messages as well.
- The locale is applicable to all conversions by the LDAP library within the application address space. The LDAP locale must be set or changed only when there is no other LDAP activity that occurs within the application on other threads.

Notes

These notes provide information that is useful in the LDAP structure.

Do not make any assumptions about the order or location of elements in the opaque LDAP structure.

See also

You can see more information about the LDAP structure in `ldap_bind`.

See ldap_bind.

LDAP_MEMFREE

Use the `LDAP_MEMFREE` API to free the storage that is allocated by the LDAP library.

```
ldap_memfree
ldap_ber_free
ldap_control_free
ldap_controls_free
ldap_msgfree
```

Synopsis

```
#include ldap.h


void ldap_memfree(
        char            *mem)

void ldap_ber_free(
        BerElement      *berptr)

void ldap_control_free (
        LDAPControl     *ctrl)

void ldap_controls_free)
        LDAPControl     **ctrls)

int ldap_msgfree(
       LDAPMessage      *msg)
```

Input parameters

mem Specifies the address of storage that is allocated by the LDAP library.

berptr Specifies the address of the `BerElement` returned from `ldap_first_attribute()` and `ldap_next_attribute()`.

ctrl Specifies the address of an `LDAPControl` structure.

ctrls Specifies the address of an `LDAPControl` list, represented as a NULL-terminated array of pointers to `LDAPControl` structures.

Usage

The `ldap_memfree()` API is used to free storage that is allocated by the LDAP library (`libldap`). Use this routine as directed when you use `ldap_get_option()`, `ldap_first_attribute()`, `ldap_default_dn_get()`, and `ldap_enetwork_domain_get()`.

The `ldap_ber_free()` API is used to free the `BerElement` pointed to by `berptr`. The LDAP library automatically frees the `BerElement` when `ldap_next_attribute()` returns NULL. The application is responsible for freeing the `BerElement` if it does not call `ldap_next_attribute()` until it returns NULL.

For those LDAP APIs that allocate an `LDAPControl` structure, the `ldap_control_free()` API can be used.

For those LDAP APIs that allocate an array of `LDAPControl` structures, the `ldap_controls_free()` API can be used.

The `ldap_msgfree()` routine is used to free the memory that is allocated for an LDAP message by `ldap_result`, `ldap_search_s`, `ldap_search_ext_s()`, or `ldap_search_st()`. It takes a pointer to the result to be freed and returns the type of the message it freed.

See also

ldap_controls

LDAP_MESSAGE

Use the `LDAP_MESSAGE` API or LDAP routine to step through the list of messages in a result chain, as returned by `ldap_result()`.

`ldap_first_message`

`ldap_next_message`

```
ldap_count_messages
```

Synopsis

```
#include ldap.h


LDAPMessage *ldap_first_message(
                LDAP           *ld,
                LDAPMessage    *result)

LDAPMessage *ldap_next_message(
                LDAP           *ld,
                LDAPMessage    *msg)

int ldap_count_messages(
                LDAP           *ld,
                LDAPMessage    *result)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or`ldap_open()`.

result Specifies the result that is returned by a call to `ldap_result()` or one of the synchronous search routines such as `ldap_search_s()`, `ldap_search_st()`, or `ldap_search_ext_s()`.

msg Specifies the message that is returned by a previous call to `ldap_first_message()` or `ldap_next_message()`.

Usage

These routines are used to step through the list of messages in a result chain, as returned by `ldap_result()`.

For search operations, the result chain can include:
- Referral messages
- Entry messages
- Result messages

The `ldap_count_messages()` API is used to count the number of messages returned. The `ldap_msgtype()` API can be used to distinguish between the different message types. Unlike `ldap_first_entry()`, `ldap_first_message()` returns any of the three types of messages.

The `ldap_first_message()` and `ldap_next_message()` APIs return NULL when no more messages exist in the result that is set to be returned. NULL is also returned if an error occurs while you step through the entries. When such an error occurs, `ldap_get_errno()` can be used to obtain the error code.

The `ldap_count_messages()` API can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference that is returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, and `ldap_next_reference()`.

Errors

If an error occurs in `ldap_first_message()` or `ldap_next_message()`, the `ldap_get_errno()` API can be used to obtain the error code.

If an error occurs in `ldap_count_messages()`, `-1` is returned, and `ldap_get_errno()` can be used to obtain the error code. See "LDAP_ERROR" on page 45 for a description of possible error codes.

See also

ldap_result, ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_get_errno, ldap_msgtype.

LDAP_MODIFY

Use the `LDAP_MODIFY` API to conduct various LDAP modify operations.

- `ldap_modify`
- `ldap_modify_ext`
- `ldap_modify_s`
- `ldap_modify_ext_s`
- `ldap_mods_free`

Synopsis

```
#include ldap.h


 typedef struct ldapmod {
        int mod_op;
        char *mod_type;
        union {
        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(
       LDAP           *ld,
       const char     *dn,
       LDAPMod        *mods[])

int ldap_modify_ext(
       LDAP           *ld,
       const char     *dn,
       LDAPMod        *mods[],
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

int ldap_modify_s(
       LDAP           *ld,
       const char     *dn,;
       LDAPMod        *mods[])

int ldap_modify_ext_s(
       LDAP           *ld,
       const char     *dn,
       LDAPMod        *mods[],
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls)
```

```
void ldap_mods_free(
       LDAPMod        **mods,
       int            *freemods)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

dn Specifies the distinguished name (DN) of the entry to be modified. For more information about DNs, see Appendix C, "LDAP distinguished names," on page 187.

mods Specifies a NULL-terminated array of entry modifications. Each element of the **mods** array is a pointer to an `LDAPMod` structure.

freemods
Specifies whether the **mods** pointer is to be freed, in addition to the NULL-terminated array of mod structures.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the `ldap_modify_ext()` call succeeds.

Usage

The various modify APIs are used to run an LDAP modify operation. DN is the distinguished name of the entry to modify, and **mods** is a NULL-terminated array of modifications to make to the entry. Each element of the **mods** array is a pointer to an `LDAPMod` structure.

The **mod_op** field is used to specify the type of modification to run and must be one of the following types:

- `LDAP_MOD_ADD (0x00)`
- `LDAP_MOD_DELETE (0x01)`
- `LDAP_MOD_REPLACE (0x02)`

This **mod_op** field also indicates the type of values that are included in the `mod_vals` union. For binary data, you must also logically or the operation type with `LDAP_MOD_BVALUES (0x80)`. This type indicates that the values are specified in a NULL-terminated array of struct berval structures. Otherwise, the `mod_values` are used, that is, the values are assumed to be a NULL-terminated array of NULL-terminated character strings.

The **mod_type** field specifies the name of the attribute to add, modify, or delete.

The mod_vals field specifies a pointer to a NULL-terminated array of values to add, modify, or delete. Only one of the mod_values or mod_bvalues variants must be used, with mod_bvalues being selected by ORing the **mod_op** field with the constant LDAP_MOD_BVALUES.

The mod_values array is NULL-terminated. Because the ldap_add() API converts the string from the local code page to UTF-8, the strings must be in the local code page if the LDAP_OPT_UTF8_IO option is set to LDAP_UTF8_XLATE_ON for the connection. If the UTF-8 translation option is not set, the array of strings must be composed of NULL-terminated UTF-8 strings.

Note: US-ASCII is a subset of UTF-8.

mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

For LDAP_MOD_ADD modifications, the values are added to the entry, creating the attribute if necessary.

For LDAP_MOD_DELETE modifications, the values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the **mod_values** field must be set to NULL.

For LDAP_MOD_REPLACE modifications, the attribute has the listed values after the modification, which is created if necessary, or removed when the **mod_vals** field is NULL.

All modifications are run in the order in which they are listed.

The ldap_modify_ext() API initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or it returns another LDAP error code if it is not successful. 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 operation. When the operation is complete, ldap_result() returns the status of the operation in the form of an error code. The error code indicates whether the operation completed successfully. The ldap_parse_result() API checks the error code in the result.

The ldap_modify() API initiates an asynchronous modify operation and returns the message ID of this operation. A subsequent call to ldap_result(), can be used to obtain the result of ldap_modify(). If there is an error, ldap_modify() returns -1, which sets the session error parameters in the LDAP structure appropriately. The parameters can be obtained by using ldap_get_errno(). For more information, see "LDAP_ERROR" on page 45.

The synchronous ldap_modify_ext_s() and ldap_modify_s() APIs 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() APIs support LDAP V3 server controls and client controls.

The ldap_modify_s() API returns the LDAP error code that results from the modify operation. This code can be interpreted by ldap_perror() or ldap_err2string().

The `ldap_modify()` operation works the same way as `ldap_modify_s()`, except that it is asynchronous, returning the message ID of the request it initiates, or `-1` on error. The result of the operation can be obtained by calling `ldap_result()`.

`ldap_mods_free()` can be used to free each element of a NULL-terminated array of `LDAPMod` structures. If `freemods` is nonzero, the **`mods`** pointer is freed as well.

Errors

`ldap_modify_s()` and `ldap_modify_ext_s()` return the resulting LDAP error code from the modify operation.

`ldap_modify()` and `ldap_modify_ext()` return `-1` instead of a valid **`msgid`** if an error occurs, setting the session error in the LD structure, which can be obtained by using `ldap_get_errno()`. For more information, see "LDAP_ERROR" on page 45.

See also

ldap_error, ldap_add

LDAP_PAGED_RESULTS

Use the `LDAP_PAGED_RESULTS` API or LDAP routine to request simple paged results of entries that are returned by the servers that match the filter that is specified on a search operation.

`ldap_create_page_control`

`ldap_parse_page_control`

Synopsis

```
#include ldap.h

int ldap_create_page_control(
   LDAP            *ld,
   unsigned long    pageSize,
   struct berval   *cookie,
   const char       isCritical,
   LDAPControl     **control)

int ldap_parse_page_control(
   LDAP            *ld,
   LDAPControl     **serverControls,
   unsigned long   *totalCount,
   struct berval   **cookie)
```

Input parameters

ld Specifies the LDAP pointer that is returned by previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`. Must not be NULL.

pageSize
Number of entries that are returned for this paged results search request.

cookie Opaque structure that is returned by the server. No assumptions must be made about the internal organization or value. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie that is returned on the last response from the server on all subsequent paged results search requests. The cookie is empty when:

- There are no more entries to be returned by the server

- The client abandons the paged results request by sending in a zero page size

After the paged results search request is completed, the cookie must not be used because it is no longer valid.

isCritical
Specifies the criticality of paged results on the search. Whether the criticality of paged results is `TRUE` or `FALSE`, and the server finds a problem with the sort criteria, the search does not continue. If the server does not find any problem with the paged results criteria, the search continues and entries are returned one page at a time.

serverControls
A list of LDAP server controls. For more information about server controls, see "LDAP controls" on page 27. These controls are returned to the client when you call the `ldap_parse_result()` function on the set of results that are returned by the server.

Output parameters

control
A result parameter that is provided with an allocated array of one control for the sort function. The control must be freed by calling `ldap_control_free()`.

totalCount
Estimate of the total number of entries for this search, can be zero if the estimate cannot be provided.

cookie Opaque structure that is returned by the server. No assumptions must be made about the internal organization or value. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie that is returned on the last response from the server on all subsequent paged results search requests. The cookie is empty when:

- There are no more entries to be returned by the server
- The client abandons the paged results request by sending in a zero page size

After the paged results search request is completed, the cookie must not be used because it is no longer valid.

Usage

The `ldap_create_page_control()` function uses the page size and the cookie to build the paged results control. The control output from `ldap_create_page_control()` function includes the criticality set that is based on the value of the `isCritical` flag. This control is added to the list of client controls that are sent to the server on the LDAP search request.

When a paged results control is returned by the server, the `ldap_parse_page_control()` function can be used to retrieve the values from the control. The function takes as input the server controls returned by the server. Then, it returns a cookie to be used on the next paged results request for this search operation.

Note: If the page size is greater than or equal to the search `sizeLimit` value, the server ignores the paged results control because the request can be satisfied in a single page. No paged results control value is returned by the server in this case. In all other cases, error or not, the server returns a paged results control to the client.

Simple paged results of search results

Simple Paged Results provides paging capabilities for LDAP clients that want to receive just a subset of search results (page) instead of the entire list. The next page of entries is returned to the client application for each subsequent paged results search request. The request is submitted by the client until the operation is canceled or the last result is returned. The server ignores a simple paged results request if the page size is greater than or equal to the `sizeLimit` value for the server because the request can be satisfied in a single operation.

The `ldap_create_page_control()` API takes as input a page size and a cookie, and outputs an `LDAPControl` structure that can be added to the list of client controls that are sent to the server on the LDAP search request. The page size specifies search results that must be returned for this request, and the cookie is an opaque structure that is returned by the server. On the initial paged results search request, the cookie must be a zero-length string. No assumptions must be made about the internal organization or value of the cookie. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie that is returned on the last response from the server on all subsequent paged results search requests. The cookie is empty when:

- There are no more entries to be returned by the server
- The client application abandons the paged results request by sending in a zero page size

After the paged results search request is completed, the cookie must not be used because it is no longer valid.

The `LDAPControl` structure that is returned by `ldap_create_page_control()` can be used as input to `ldap_search_ext()` or `ldap_search_ext_s()`, which are used to make the actual search request.

Note: Server side simple paged results is an optional extension of the LDAP v3 protocol, so the server you bound before the `ldap_search_ext()` or `ldap_search_ext_s()` call might not support this function.

Upon completion of the search request that you submitted by using `ldap_search_ext()` or `ldap_search_ext_s()`, the server returns an LDAP result message that includes a paged results control. The client application can parse this control by using `ldap_parse_page_control()`, which takes the returned server response controls (a null terminated array of pointers to `LDAPControl` structures) as input. `ldap_parse_page_control()` outputs a cookie and the total number of entries in the entire search result-set. Servers that cannot provide an estimate for the total number of entries might set this value to zero. Use `ldap_controls_free()` to free the memory that is used by the client application to hold the server controls when you are finished processing all controls that are returned by the server for this search request.

The server might limit the number of outstanding paged results operations from a client or for all clients. A server with a limit on the number of outstanding paged results requests might return either `LDAP_UNWILLING_TO_PERFORM` in the `sortResultsDone` message or age out an older paged results request. There is no surety to the client application that search query results remain unchanged throughout the life of a paged results request set or response sequences. The result-set for that query might changes since the initial search request by specifying paged results. If it changes, the client application might not receive all the entries that match the search criteria. When you chase referrals, the client application must send an initial paged results request, with the cookie set to null, to each of the referral servers. It is up to the application that uses the client services to decide whether to set the criticality as to the support of paged results. It is also up to the application to handle a lack of support of this control on referral servers as appropriate, based on the application. Additionally, the LDAP server does not ensure that the referral server supports the paged results control. Multiple lists can be returned to the client application, some not paged. It is the client decision of the application about how best to present this information to the user. Possible solutions include:

- Combine all referral results before you present to the user
- Show multiple lists and the corresponding referral server host name
- Take no extra steps and show all results to the user as they are returned from the server

The client application must turn off referrals to get one truly paged list. Otherwise, when you chase referrals with the paged results search control specified, unpredictable results might occur.

Information about simple paged results search control with control OID of 1.2.840.113556.1.4.319 can be found in the RFC 2696, LDAP Control Extension for Simple Paged Results Manipulation.

Errors

The sort routines return an LDAP error code if they encounter an error that parses the result. For a list of the LDAP error codes, see "LDAP_ERROR" on page 45.

Notes

Controls, `serverControls`, and cookie must be freed by the caller.

See also

ldap_search, ldap_parse_result

LDAP_PARSE_EFFECTIVE_PWDPOLICY_RESPONSE

Use the `LDAP_PARSE_EFFECTIVE_PWDPOLICY_RESPONSE` API or LDAP routine for extracting information from results that are returned by the `ldap_parse_effective_pwdpolicy_request()` routine.

Synopsis

```
#include ldap.h


int ldap_parse_effective_pwdpolicy_response(
                         char            *resOid,
                         struct berval   **resVal,
                         LDAPAttr        **attrs,
                         char            **dns);
```

Input parameters

resOid
Represents the effective password policy response OID.

resVal Specifies a berval structure that contains the response value of the effective password policy extended operation. This **resVal** must be an output of `ldap_extended_operation` or `ldap_extended_operation_s` function.

Output parameters

attrs Specifies an array of pointers that point to structures that stores the requested user or group effective password policy attribute types and values.

dns Specifies an array of pointers that point to entry DNs from which the requested effective password policy is derived.

Usage

This API is used to obtain the attribute types and attribute values that are contained in the returned response of an effective password policy extended operation. If the bind DN of this extended operation is an administrative user, the DN of the password policy entries is also returned. The DN of the password policy entries is evaluated with the effective password policy.

Errors

The `ldap_parse_effective_pwdpolicy_response` routine returns an LDAP error code if it encounters an error in parsing the result.

See also

ldap_create_effective_pwdpolicy_request, ldap_extended_operation, ldap_extended_operation_s

LDAP_PARSE_ENTRYCHANGE_CONTROL

The `LDAP_PARSE_ENTRYCHANGE_CONTROL` API or LDAP routine is used by a client application to parse the control when the client application receives entries that contain Entry Change Notification controls.

Synopsis

```
#include ldap.h


#define LDAP_CONTROL_ENTRYCHANGE   "2.16.840.1.113730.3.4.7"

int ldap_parse_entrychange_control(
           LDAP           *ld,
           LDAPControl    **ctrls,
           int            *chgtypep,
           char           **prevdnp,
           int            *chgnumpresentp,
           long           *chgnump);
```

Input parameters

ld Specifies the LDAP pointer, which acts as an LDAP session handle, returned by previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

ctrls Specifies the address of a NULL-terminated array of `LDAPControl` structures that are obtained by calling `ldap_get_entry_controls()`.

Output parameters

chgtypep
This result parameter contains the value that indicates the type of change made that caused the entry to be returned. The value for this result parameter is obtained from the `changeType` element of the BER-encoded `EntryChangeNotification` control value. The parameter can contain one of the following values:

- `LDAP_CHANGETYPE_ADD (1)`
- `LDAP_CHANGETYPE_DELETE (2)`
- `LDAP_CHANGETYPE_MODIFY (4)`
- `LDAP_CHANGETYPE_MODDN (8)`
- `NULL`

If this parameter is NULL, the change type information is not returned.

prevdnp
This result parameter is provided with the DN that an entry had before the DN was renamed or moved by the `modifyDN` operation. For other type of changes, the value of parameter is set to NULL. If the parameter is NULL, the previous DN information is not returned. The value for this result parameter is pulled from the `previousDN` element of the BER-encoded `EntryChangeNotification` control value.

chgnumpresentp
This result parameter contains a non-zero value if a change was returned in the `EntryChangeNotification` control. If this parameter is NULL, there is no indication whether the change number was present.

Note: Even if the parameter contains a non-zero value, the server might choose not to return the change number because it is optional.

chgnump
This result parameter contains the change number if a change was returned in the `EntryChangeNotification` control. If this parameter contains a non-NULL value, the **`chgnumpresentp`** parameter is provided with a non-zero value. If this parameter is NULL, the change number is not returned. The value for this result parameter is pulled from the optional `changeNumber` element of the BER-encoded `EntryChangeNotification` control value.

Usage

This routine is used by a client application to search and parse the control when the client application receives entries that contain Entry Change Notification controls. If the operation is successful, `LDAP_SUCCESS` is returned.

Errors

This routine returns LDAP error code that indicates whether an `EntryChangeNotification` control was found and the parsing was successful. `LDAP_CONTROL_NOT_FOUND` is returned if the **`ctrls`** array does not include an `EntryChangeNotification` control.

LDAP_PARSE_EXTENDED_RESULT_W_CONTROLS

Use the `LDAP_PARSE_EXTENDED_RESULT_W_CONTROLS` API or LDAP routine to parse the extended response result and return any response controls that are sent on the response.

Synopsis

```
#include ldap.h


int ldap_parse_extended_result_w_controls (
          LDAP                 *ld,
          LDAPMessage          *res,
          char                 **resultoidp,
          struct berval        **resultdata,
          int                  freeit,
          LDAPControl          ***serverctrlsp);
```

Input parameters

ld Specifies a pointer to the LDAP structure that represents an LDAP connection.

res Specifies a pointer to `LDAPMessage` structure that points to the result of the operation that is returned by `ldap_result()`.

resultoidp
A character pointer specifies the location of the dotted-OID text that represents the name of the extended operation. A NULL value can be passed to this parameter.

resultdata
A pointer to the struct berval that points to the data in the extended operation response. A NULL value can be passed to this parameter.

freeit An integer variable, which indicates whether the result parameter must be freed after the information is extracted.

Output parameters

serverctrlsp
Specifies a pointer to a result parameter that is provided with an allocated array of controls that are copied out of the `LDAPMessage` structure. The control array must be freed by calling `ldap_controls_free()`.

LDAP Return code
The return code is set as listed for `ldap_parse_extended_result`.

Usage

The `ldap_parse_extended_result_w_controls()` API is used after you call `ldap_extended_operation` to get the results. This routine must be called if controls are returned as part of the extended operation result.

LDAP_PARSE_LIMIT_NUM_VALUES_RESPONSE

Use the `LDAP_PARSE_LIMIT_NUM_VALUES_RESPONSE` API or LDAP routine to extract information from the results that are returned by the Limit Number of Attribute Values Control.

Synopsis

```
 #include ldap.h

    int ldap_parse_limit_num_values_response(
            LDAP                  *ld,
            LDAPCONTROL           **serverControls,
            LDAPNumValuesResponse  **numValuesResponse);
```

Input parameters

ld Specifies a pointer to the LDAP structure that represents an LDAP connection.

serverControls
Specifies an array of **LDAPCONTROL** pointers that are returned by a previous call to `ldap_parse_result()`.

numValuesResponse
Specifies the address of a pointer to an `LDAPNumValuesResponse` structure in which the control results are placed.

Usage

The `ldap_parse_limit_num_values_response` routine is used for obtaining the results of the Limit Number of Attribute Values Control that was used in a search operation. The results are built into an `LDAPNumValuesResponse` structure.

Note: The `LDAPNumValuesResponse` structure that is created by this routine must be freed by calling the `ldap_free_limit_num_values_response()` API.

Errors

The errors that are returned by the `ldap_parse_limit_num_values_response` routine are listed:

- `LDAP_SUCCESS` // is returned if the operation is successful
- `LDAP_DECODING_ERROR` // is returned if the response cannot be parsed
- `LDAP_PARAM_ERROR` // is returned if an input parameter is not valid
- `LDAP_NO_MEMORY` // is returned if the server runs out of memory
- `LDAP_OPERATIONS_ERROR` //is returned if any other internal error occurs

See also

ldap_parse_result, ldap_free_limit_num_values_response

LDAP_PARSE_RESULT

Use the `LDAP_PARSE_RESULT` API or LDAP routine to extract information from the results that are returned by other LDAP API routines.

```
ldap_parse_result
ldap_parse_sasl_bind_result
ldap_parse_extended_result
```

Synopsis

```
#include ldap.h


int ldap_parse_result(
      LDAP           *ld;
```

```
       LDAPMessage     *res,
       int             *errcodep,
       char            **matcheddnp,
       char            **errmsgp,
       char            ***referralsp,
       LDAPControl     ***servctrlsp,
       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            **resultoidp,
       struct berval   **resultdatap,
       int             freeit)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

res
Specifies the result of an LDAP operation as returned by `ldap_result()` or one of the synchronous LDAP API operation calls.

errcodep
Specifies a pointer to the result parameter that is provided with the **LDAP error code** field from the `LDAPMessage` message. The `LDAPResult` message is produced by the LDAP server, and indicates the outcome of the operation. NULL can be specified for *errcodep* if the `LDAPResult` message is to be ignored.

matcheddnp
Specifies a pointer to a result parameter. When `LDAP_NO_SUCH_OBJECT` is returned as the LDAP error code, this result parameter is provided with a Distinguished Name indicating how much of the name in the request was recognized by the server. NULL can be specified for `matcheddnp` if the matched DN is to be ignored. The matched DN string must be freed by calling `ldap_memfree()`.

errmsgp
Specifies a pointer to a result parameter that is provided with the contents of the error message from the `LDAPMessage` message. The error message string must be freed by calling `ldap_memfree()`.

referralsp
Specifies a pointer to a result parameter that is provided with the contents of the referrals field from the `LDAPMessage` message, indicating zero or more alternative LDAP servers where the request must be tried again. The referrals array must be freed by calling ldap_value_free(). NULL can be supplied for this parameter to ignore the referrals field.

resultoidp
This result parameter specifies a pointer that is set to point to an allocated, dotted-OID text string that is returned from the server. This string must be disposed of using the `ldap_memfree()` API. If no OID is returned, **`*resultoidp`** is set to NULL.

resultdatap
This result parameter specifies a pointer to a berval structure pointer that is set to an allocated data copy that is returned by the server. This struct berval must be disposed of using ber_bvfree(). If no data is returned, `*resultdatap` is set to NULL.

serverctrlsp
Specifies a pointer to a result parameter that is provided with an allocated array of controls that are copied out of `LDAPMessage`. The control array must be freed by calling `ldap_controls_free()`.

freeit Specifies a Boolean value that determines whether the LDAP result (as specified by res) is to be freed. Any nonzero value results in **res** being freed after the requested information is extracted. The `ldap_msgfree()` API can be used to free the result later.

servercredp
Specifies a pointer to a result parameter. For SASL bind results, this result parameter is provided with the credentials returned by the server for mutual authentication, if the credentials are returned. The credentials are returned in a struct berval structure. NULL might be supplied to ignore this field.

err Specifies an LDAP error code, which is used as input to `ldap_err2string()`, so that a text description of the error can be obtained.

Usage

The `ldap_parse_result()` API is used to:
- Obtain the **LDAP error code** field that is associated with an `LDAPMessage` message.
- Obtain the portion of the DN that the server recognizes for a failed operation.
- Obtain the text error message that is associated with the error code returned in an `LDAPMessage` message.
- Obtain the list of alternative servers from the referrals field.
- Obtain the array of controls that can be returned by the server.

The `ldap_parse_sasl_bind_result()` API is used to obtain server credentials, as a result of an attempt to run mutual authentication.

Both the `ldap_parse_sasl_bind_result()` and the `ldap_parse_extended_result()` APIs ignore messages of type `LDAP_RES_SEARCH_ENTRY` and `LDAP_RES_SEARCH_REFERENCE` when you look for a result message to parse. They both return `LDAP_SUCCESS` if the result was successfully located and parsed, and an LDAP error code if the result was not successfully parsed.

The `ldap_err2string()` API is used to convert the numerical LDAP error code, as returned by any of the LDAP APIs, into a NULL-terminated character string that describes the error. The character string is returned as static data and must not be freed by the application.

Errors

The parse routines return an LDAP error code if they encounter an error that parses the result.

See “LDAP_ERROR” on page 45 for a list of the LDAP error codes.

See also

ldap_error, ldap_result

LDAP_PARSE_VLV_CONTROL

Use the `LDAP_PARSE_VLV_CONTROL` API or LDAP routine to parse a virtual list view control.

Synopsis

```
#include ldap.h

int ldap_parse_vlv_control(
 LDAP           *ld,
 LDAPControl    **ctrlp,
 unsigned long  *target_posp,
 unsigned long  *list_countp,
 struct berval  **contextp,
 int            *errcodep
);
```

Input parameters

ld Specifies the LDAP session handle that is returned by a call to `ldap_init()`.

ctrlp Contains the address of NULL terminated array of `LDAPControl` structures, typically obtained by a call to `ldap_parse_result()`.

target_posp
Specifies the result parameter that contains the list index of the target entry. If the value of this parameter is NULL, then the target position is not returned. The value for this result parameter is obtained from the **targetPosition** field in the Virtual list view response control.

list_countp
Specifies the result parameter that contains the server estimate of the list size. If the value of this parameter is NULL, then the size is not returned. The value for this result parameter is obtained from the **contentCount** field of the Virtual list view response control.

contextp
Specifies the result parameter that contains the address of struct berval, which contains a server generated context ID if one was returned by the server. If the server did not return any context ID, the value of the parameter is NULL. The berval structure that returned must be freed after its use by using the `ber_bvfree()` function.

errcodep
Specifies the result parameter that contains the Virtual list view result code. If NULL, the result code is not returned. The value of this parameter is obtained from the **virtualListViewResult** field of the Virtual list view response control.

Usage

The `ldap_parse_vlv_control` routine is used to parse the virtual list view response control. The following codes are the possible return codes from this routine:

- `LDAP_SUCCESS` // on successful parsing
- `LDAP_NO_MEMORY` // memory allocation failure

- `LDAP_PARAM_ERROR` // bad input parameters
- `LDAP_DECODING_ERROR` // decoding error

See also

ldap_create_vlv_control

LDAP_PASSWORD_POLICY

Use the `LDAP_PASSWORD_POLICY` API or LDAP routine to extract information from results that are returned in the Password Policy Control structure.

`ldap_parse_pwdpolicy_response`

`ldap_pwdpolicy_err2string`

Synopsis

```
#include ldap.h

int ldap_parse_pwdpolicy_response(LDAPCONTROL **serverControls,
       int *controlerr,
       int *controlwarn,
       int *controlres);

static struct ldaperror ldap_ctrlerr[] = {
    LDAP_SUCCESS,                     "Success",
    LDAP_TIME_BEFORE_EXPIRE,          "Warning, time before expiration is %ld",
    LDAP_GRACE_LOGINS,                "Warning, %ld grace logins remain",
    LDAP_PASSWORD_EXPIRED,            "Error, Password has expired",
    LDAP_ACCOUNT_LOCKED,              "Error, Account is locked",
    LDAP_CHANGE_AFTER_RESET,          "Error, Password must be changed after reset",
    LDAP_PASSWORD_NO_MOD,             "Error, Password may not be modified",
    LDAP_NEED_OLD_PASSWORD,           "Error, Must supply old password",
    LDAP_INVALID_PASS_SYNTAX,         "Error, Invalid password syntax",
    LDAP_PASSWORD_TOO_SHORT,          "Error, Password too short",
    LDAP_PASSWORD_TOO_YOUNG,          "Error, Password too young",
    LDAP_PASSWORD_IN_HISTORY,         "Error, Password in History",
    -1, NULL
};
const char *ldap_pwdpolicy_err2string(int err);
```

Input parameters

serverControls
: Specifies an array of `LDAPCONTROL` pointers that are returned by a previous call to `ldap_parse_result()`.

controlerr
: Specifies a pointer to the result parameter that is provided with the LDAP Password Policy error code, which can be used as input to `ldap_pwdpolicy_err2string()`, so that a text description of the error can be obtained.

controlwarn
: Specifies a pointer to the result parameter that is provided with the LDAP Password Policy warning code, which can be used as input to `ldap_pwdpolicy_err2string()`, so that a text description of the warning can be obtained.

controlres
: Specifies a pointer to the result parameter that is provided with the LDAP Password Policy warning result value.

err Specifies an integer value that is returned from `ldap_parse_pwdpolicy_response()` that contains the Password Policy warning or error code.

Usage

The `ldap_parse_pwdpolicy_response()` API is used to:

- Obtain the LDAP Password Policy error or warning codes from the Password Policy Response Control that is associated with an `LDAPMessage` message.
- Obtain the LDAP Password Policy warning result code from the Password Policy Response Control that is associated with the returned Password Policy warning code.
- This function takes in an array of `LDAPCONTROL` structure pointers, parses these structures, and then returns three integers that contain the Password Policy response values.

The static struct `ldaperror` **`ldap_ctrlerr`** array contains the LDAP error code that is associated with the password policy and the corresponding text description of the LDAP error code. Unlike the familiar `ldap_error2string`, for warnings, you get a string with `%d` format, and pass it to the **`printf()`** function along with the returned **`controlres`** to get the final diagnostic string. In some cases, the text description string has `%ld`, and expects some parameters before it can be printed.

The `ldap_pwdpolicy_err2string()` API is used to convert the numerical LDAP Password Policy error or warning code, as returned by `ldap_parse_pwdpolicy_response()`, into a NULL-terminated character string that describes the error or warning. The character string is returned as static data and must not be freed by the application.

Errors

The `ldap_parse_pwdpolicy_response` routine returns an LDAP error code if it encounters an error that parses the result.

See "LDAP_ERROR" on page 45 for a list of the LDAP error codes.

See also

ldap_parse_result

LDAP_PLUGIN_REGISTRATION

Use the `LDAP_PLUGIN_REGISTRATION` API or LDAP routine to register an LDAP client plug-in, obtain information about plug-ins that are registered by the application and plug-ins that are defined in `ibmldap.conf`, and free the array of plug-in information that is returned from the `ldap_query_plugin()` API.

```
ldap_register_plugin
ldap_query_plugin
ldap_free_query_plugin
```

Synopsis

```
#include ldap.h


int ldap_register_plugin(
       LDAP_File_Plugin_Info *plugin_info)

int ldap_query_plugin(
       LDAP_File_Plugin_Info  plugin_infop )

int ldap_free_query_plugin(
       LDAP_File_Plugin_Info  ***plugin_infop )

typedef struct ldap_file_plugin_info {
    char    *type;             /* plug-in type               */
    char    *subtype;          /* plug-in subtype            */
    char    *path;             /* path to plug-in library    */
    char    *init;             /* initialization routine    */
    char    *paramlist;        /* plug-in parameter list     */
} LDAP_File_Plugin_Info;
```

Input parameters

plugin_info
A structure that contains information about a specific type of SASL plug-in. An instance of the structure contains the following fields:

type NULL-terminated string that defines the plug-in type. The only type that is supported is `sasl`.

subtype
NULL-terminated string that specifies the subtype of the plug-in being registered. When `type=sasl`, the subtype is used to specify the SASL mechanism that is supported by the plug-in. For example, fingerprint might be specified for any SASL plug-in that supports the fingerprint mechanism.

path NULL-terminated string that specifies the path to the plug-in shared library. The plug-in path can be a fully qualified path that includes file name, or only the file name with or without the file extension. If only the file name is supplied, the LDAP library attempts to find it using standard operating system search criteria.

init NULL-terminated string that specifies the initialization routine for the plug-in. If NULL, the name of the initialization routine is assumed to be `ldap_plugin_init`.

parmlist
NULL-terminated string that specifies arbitrary parameter information that is used by the plug-in. For example, if the plug-in accesses a remote security server, then the remote security server host name is supplied as a value in the parameter list.

plugin_infop
Specifies the address that points to a NULL-terminated array of `LDAP_Plugin_Info` structures. Each `LDAP_Plugin_Info` structure that is defined in the list contains information about a registered plug-in. For example:

```
LDAP_File_Plugin_Info  **plugin_infop;

             rc = ldap_query_plugin (&plugin_infop);
```

Output parameters

plugin_infop
Upon successful return from `ldap_query_plugin()`, `plugin_infop` points to

a NULL-terminated array of `LDAP_Plugin_Info` pointers. If there are no plug-ins that are registered, the `plugin_infop` data structure is set to NULL and no memory is allocated.

Usage

Two mechanisms are available for making an LDAP client plug-in that is known to the LDAP library:

- The plug-in is defined in the `ibmldap.conf` file.
- The plug-in is explicitly registered by the application, by using the `ldap_register_plugin()` API.

An application can override the definition of a plug-in in the `ibmldap.conf` file by using the `ldap_register_plugin()` API. A plug-in is uniquely identified by the combination of its type and subtype. For example, an application can choose to use its own `DIGEST-MD5` plug-in as defined in `ibmldap.conf`, by calling `ldap_register_plugin()` and defining another shared library with `type="sasl"` and `subtype="DIGEST-MD5"`. Plug-ins that are registered with the `ldap_register_plugin()` API are defined for the application.

Finding the plug-in library

When a plug-in is not explicitly registered by the application with the `ldap_register_plugin()` API, the LDAP library must find the appropriate plug-in shared library. To find information about the plug-in, the LDAP library must find the `ibmldap.conf` file. The attempt to locate the `ibmldap.conf` file is made on behalf of the application in whichever of the following events occurs first:

- The `ldap_register_plugin()` API is called.
- The `ldap_sasl_bind_s()` API is called.

After the `ibmldap.conf` file is accessed, all information in the file is stored internally for subsequent use. The file is not reaccessed until the application is restarted. However, the application can use the `ldap_register_plugin()` API to add more plug-in definitions, or to override definitions that are obtained from the `ibmldap.conf` file.

The `ibmldap.conf` file

The `ibmldap.conf` file contains information that is required to load and initialize default plug-ins. It can also include more plug-in-specific configuration information. The following types might be defined for each plug-in in the `ibmldap.conf` file:

- The plug-in type (for example, `sasl`)
- The plug-in subtype (for example, mechanism, if `type=sasl`)
- The path to the plug-in shared library
- The plug-in initialization routine
- The user-defined parameter string

The `ibmldap.conf` file might contain one or more records, each defining this information for a plug-in. Each record takes the following form:

```
plugin type  subtype  path  init-routine parameters
```

For example:

```
#
#   keyword type   subtype          path              init      parameters
#
    plugin  sasl   fpauth    x:\security\fplib        fpinit    parm2 parm3
    plugin  sasl   hitech    hitechlib                hitekinit parm5 parm6
```

This example defines two plug-ins, `fpauth` and `hitek`, along with associated information.

Note: If the extension is omitted, then an appropriate extension is assumed for the platform. For example, `.a` on the AIX operating system or `.dll` on a Windows operating system. If the fully qualified path is omitted, standard operating system search rules are applied.

Lines beginning with a number sign (#) are ignored.

The algorithm that is used to locate the `ibmldap.conf` file is platform-specific:

- On a UNIX system, the following search order is used:
  1. Query the environment variable *IBMLDAP_CONF* for the path to the `ibmldap.conf` file.
  2. Look for the `ibmldap.conf` file in the `/etc` directory.
- On a Windows system, the following search order is used:
  1. Query the environment variable *IBMLDAP_CONF* for the path to the `ibmldap.conf` file.
  2. Look in the current directory for the `ibmldap.conf` file.
  3. Look for the `ibmldap.conf` file in the `\etc` directory under the LDAP installation directory. For example, `C:\Program Files\IBM\ldap\V6.3.1\etc`.

If the SASL plug-in definition is not available, the LDAP library encodes the SASL bind. It transmits it directly to the LDAP server and bypasses the plug-in facility.

Errors

These routines return an LDAP error code when an error is encountered. To obtain a string description of the LDAP error, use the `ldap_err2string()` API.

See also

ldap_error

LDAP_PREPARE_TRANSACTION

Use the `LDAP_PREPARE_TRANSACTION` API or LDAP routine to call a prepare transaction request.

- `ldap_prepare_transaction`
- `ldap_prepare_transaction_s`

Synopsis

```
#include ldap.h


int ldap_prepare_transaction(
       LDAP          *ld,
       string        tran_id,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

int ldap_prepare_transaction_s(
       LDAP           *ld,
       string         tran_id,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

tran_id
Specifies the transaction ID of a prepare transaction.

serverctrls
Specifies a list of LDAP server controls.

clientctrls
Specifies a list of LDAP client controls.

Output parameters

msgidp
This parameter contains the message ID of the request.

Usage

This API routine is used to initiate a prepare transaction request against the server.

Errors

This routine returns an LDAP error code if the operation is unsuccessful.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control

LDAP_RENAME

Use the `LDAP_RENAME` API to carry out an LDAP rename operation.

`ldap_rename`

`ldap_rename_s`

`ldap_modrdn`

`ldap_modrdn_s`

Synopsis

```
#include ldap.h


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)

int ldap_modrdn(
       LDAP          *ld,
       const char    *dn,
       const char    *newrdn,
       int           deleteoldrdn)

int ldap_modrdn_s(
       LDAP          *ld,
       const char    *dn,
       const char    *newrdn,
       int           deleteoldrdn)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

dn Specifies the DN of the entry whose DN is to be changed. When specified with the `ldap_modrdn()` and `ldap_modrdn_s()` APIs, dn specifies the DN of the entry whose RDN is to be changed.

newrdn
Specifies the new RDN given to the entry.

newparent
Specifies the new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is changed. The root DN can be specified by passing a zero length string, `""`. The **`newparent`** parameter is always NULL when you use version 2 of the LDAP protocol. Otherwise, the behavior of the server is undefined.

deleteoldrdn
Specifies an integer value. When set to 1, the old RDN value is to be deleted from the entry. When set to 0, the old RDN value must be retained as a non-distinguished value. Regarding the `ldap_rename()` and `ldap_rename_s()` APIs, this parameter has meaning only if **`newrdn`** is different from the old RDN.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see “LDAP controls” on page 27.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see “LDAP controls” on page 27.

Output parameters

msgidp
: This result parameter is set to the message ID of the request if the `ldap_rename()` call succeeds.

Usage

In LDAP V2, the `ldap_modrdn()` and `ldap_modrdn_s()` APIs were used to change the name of an LDAP entry. They can be used to change the least significant component of a name (the RDN or relative distinguished name) only. LDAP V3 provides the Modify DN protocol operation that allows more general name change access. The `ldap_rename()` and `ldap_rename_s()` routines are used to change the name of an entry.

The `ldap_rename()` API 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 message ID of the request in **msgidp*. A subsequent call to `ldap_result()` can be used to obtain the result of the operation. After the operation completes, `ldap_result()` returns the status of the operation in the form of an error code. The error code indicates whether the operation completed successfully. The `ldap_parse_result()` API is used to check the error code in the result.

Similarly, the `ldap_modrdn()` API initiates an asynchronous modify RDN operation and returns the message ID of the operation. A subsequent call to `ldap_result()` can be used to obtain the result of the modify operation. If there is an error, `ldap_modrdn()` returns `-1`, setting the session error parameters in the LDAP structure appropriately, which can be obtained by using `ldap_get_errno()`.

The synchronous `ldap_rename_s()` API 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()` APIs both support LDAP V3 server controls and client controls.

The `ldap_modrdn()` and `ldap_modrdn_s()` routines run an LDAP modify RDN operation. They both take `dn`, the DN of the entry whose RDN is to be changed, and **`newrdn`**, the new RDN to give to the entry. `ldap_modrdn_s()` is synchronous, returning the LDAP error code that indicates the success or failure of the operation. In addition, they both take the **`deleteoldrdn`** parameter, which is used as an integer value to indicate whether the old RDN values must be deleted from the entry.

Errors

The synchronous version of this routine returns an LDAP error code, either `LDAP_SUCCESS` or an error code if there was an error. The asynchronous version returns `-1` in case of an error. If the asynchronous API is successful, `ldap_result()` is used to obtain the results of the operation. See "LDAP_ERROR" on page 45 for more details.

See also

ldap_error ldap_result

LDAP_RESTORE

Use the LDAP_RESTORE API or LDAP routine to create and call an LDAP extended operation that requests a directory server restore.

Synopsis

```
#include ldap.h

    int ldap_restore (LDAP *ld, , Backup_Restore_Result *op_result );
```

Input parameters

ld Specifies the address of the LDAP connection.

op_result
Specifies the address of the result code from the administration server response.

Usage

The **ldap_restore** routine is a wrapper that is used for creating requests to restore a directory server. This extended operation is only supported by the administration server, ibmdiradm.

If LDAP_SUCCESS is returned for a restore request, this status indicates that the request was sent to the administration server. The administration server submits the restore request unless the directory server is running or another backup or restore command is already running. The **op_result** parameter indicates the status of the request that is based on the action that is taken by the administration server on the request. The result of directory restore operation does not reflect in the return code or in the **op_result** parameter.

Errors

The ldap_restore routine returns the following error code:

- LDAP_SUCCESS // if the request is submitted
- LDAP_NO_MEMORY // if allocation fails
- LDAP_OPERATIONS_ERROR // if no backup is available
- LDAP_INSUFFICIENT_ACCESS // DN used for bind does not have authority to send this request
- LDAP_UNWILLING_TO_PERFORM // if backup configuration is not configured for the server
- LDAP_PROTOCOL_ERROR // if request sent to the server is from other than administration server
- LDAP_OTHER // unable to prepare request

See also

ldap_backup

LDAP_RESULT

Use the LDAP_RESULT API to wait for the result of an asynchronous LDAP operation, obtain LDAP message types, or obtain the message ID of an LDAP message.

ldap_result

ldap_msgtype

ldap_msgid

Synopsis

```
#include sys/time.h /* for struct timeval definition */
#include ldap.h


int ldap_result(
       LDAP             *ld,
       int              msgid,
       int              all,
       struct timeval   *timeout,
       LDAPMessage      **result)

int ldap_msgtype(
       LDAPMessage      *msg)

int ldap_msgid(
       LDAPMessage      *msg)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to ldap_init(), ldap_ssl_init(), or ldap_open().

msgid Specifies the message ID of the operation whose results are to be returned. The parameter can be set to LDAP_RES_ANY if any result is wanted.

all This parameter has meaning only for search results. For search results, use all to specify how many search result messages are returned in a single call to ldap_result(). Specify LDAP_MSG_ONE to retrieve one search result message at a time. Specify LDAP_MSG_ALL to request that all results of a search be received. ldap_result() waits until all results are received before it returns all results in a single chain. Specify LDAP_MSG_RECEIVED to indicate that all results retrieved so far are to be returned in the result chain.

timeout
Specifies how long in seconds to wait for results to be returned from ldap_result, as identified by the supplied **msgid**. A NULL value causes ldap_result() to wait until results are available. To poll, the timeout parameter is non-NULL, pointing to a zero-valued **timeval** structure.

msg Specifies a pointer to a result, as returned from ldap_result(), ldap_search_s(), ldap_search_st(), or ldap_search_ext().

Output parameters

result Contains the result of the asynchronous operation that is identified by **msgid**. This result is passed to an LDAP parsing routine such as ldap_first_entry().

If ldap_result() is unsuccessful, it returns -1 and sets the appropriate LDAP error, which can be retrieved by using ldap_get_errno(). If ldap_result() times out, it returns 0. If successful, it returns one of the following result types:

```
#define LDAP_RES_BIND             0x61L
#define LDAP_RES_SEARCH_ENTRY     0x64L
#define LDAP_RES_SEARCH_RESULT    0x65L
#define LDAP_RES_MODIFY           0x67L
#define LDAP_RES_ADD              0x69L
#define LDAP_RES_DELETE           0x6bL
#define LDAP_RES_MODRDN           0x6dL
#define LDAP_RES_COMPARE          0x6fL
#define LDAP_RES_SEARCH_REFERENCE 0X73L
#define LDAP_RES_EXTENDED         0X78L
#define LDAP_RES_ANY              (-1L)
#define LDAP_RES_RENAME   LDAP_RES_MODRDN
```

Usage

The ldap_result() routine is used to wait for and return the result of an operation that is previously initiated by one of the LDAP asynchronous operation routines. For example, ldap_search(), ldap_modify(), and others. These routines return a **msgid** that uniquely identifies the request. The **msgid** can then be used to request the result of a specific operation from ldap_result().

The ldap_msgtype() API returns the type of LDAP message, which is based on the LDAP message that is passed as input by using the **msg** parameter.

The ldap_msgid() API returns the message ID associated with the LDAP message passed as input by using the **msg** parameter.

Errors

ldap_result() returns 0 if the timeout expires, and -1 if an error occurs. The ldap_get_errno() routine can be used to get an error code.

Notes

This routine allocates memory for results that it receives. The memory can be deallocated by calling ldap_msgfree().

See also

ldap_search

LDAP_SEARCH

Use the LDAP_SEARCH API for carrying out various LDAP search operations.

ldap_search

ldap_search_s

ldap_search_ext

ldap_search_ext_s

ldap_search_st

Synopsis

```
#include sys/time.h /* for struct timeval definition */
#include ldap.h


int ldap_search(
       LDAP           *ld,
       const char     *base,
       int            scope,
       const char     *filter,
       char           *attrs[],
       int            attrsonly)

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_s(
       LDAP           *ld,
       const char     *base,
       int            scope,
       const char     *filter,
       char           *attrs[],
       int            attrsonly,
       LDAPMessage    **res)

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_st(
       LDAP           *ld,
       const char     *base,
       int            scope,
       const char     *filter,
       char           *attrs[],
       int            attrsonly,
       struct timeval *timeout,
       LDAPMessage    **res)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

base Specifies the DN of the entry the search starts.

scope Specifies the scope of the search. It can be `LDAP_SCOPE_BASE` (to search the object itself), or `LDAP_SCOPE_ONELEVEL` (to search the immediate children of the object), or `LDAP_SCOPE_SUBTREE` (to search the object and all its descendants).

filter
Specifies a string representation of the filter to apply in the search. Simple filters can be specified as `attributetype=attributevalue`. More complex filters are specified by using a prefix notation according to the following BNF:

```
filter ::='('filtercomp')'
filtercomp ::= and|or|not|simple
and ::= '&' filterlist
or ::= '|' filterlist
not ::= '!' filter
filterlist ::= filter|filterfiltertype
simple ::= attributetypefiltertype
attributevalue
filtertype ::= '='|'~='|'='|'='
```

The `'~='` construct is used to specify approximate matching. The representation for *attributetype* and *attributevalue* are as described in "RFC 2252, LDAP V3 Attribute Syntax Definitions". In addition, *attributevalue* can be a single * to achieve an attribute existence test, or can contain text and asterisks `( * )` interspersed to achieve substring matching.

For example, the filter `"(mail=*)"` finds any entries that have a mail attribute. The filter `"(mail=*@student.of.life.edu)"` finds any entries that have a mail attribute which ends in the specified string. To put parentheses in a filter, escape them with a backslash `( \ )` character. See "RFC 2254, A String Representation of LDAP Search Filters" for a complete description of allowable filters.

attrs
Specifies a NULL-terminated array of character string attribute types to return from entries that match filter. If NULL is specified, all user attributes are returned.

The **`attrs`** parameter consists of an array of attribute type names to be returned for each entry that matches the search filter. By default, a search request returns only user attributes. Operational attributes, for example **`createtimestamp`** and **`modifytimestamp`**, are returned only when provided in the **`attrs`** parameter. The following attributes types that are listed when specified in the **`attrs`** parameter have special meaning in LDAP searches and can be combined with other attribute types.

* Returns all user attributes.

1.1 Specifies to return no attributes and is used to request that a search returns only the matching distinguished names.

+ Returns all operational attributes.

+ibmaci
Returns the access control related operational attributes that exclude those attributes that are expensive to return.

+ibmentry
Returns a core-set of operational attributes that all entries in the directory server contain, such as **`creatorsName`** and **`createTimestamp`**. This server excludes those attributes that are expensive to return.

+ibmrepl
Returns operational attributes that are related to replication that excludes those attributes that are expensive to return.

+ibmpwdpolicy
Returns operational attributes that are related to password policy that excludes those attributes that are expensive to return.

++ Returns all operational attributes, including those attributes that are considered expensive to return, such as **`ibm-allGroups`** and **`ibm-replicationPendingChanges`**.

++ibmaci
Returns all access control related operational attributes.

++ibmentry
Returns all operational attributes that every entry contains, such as **`numsubordinates`** and **`ibm-entryChecksum`**.

++ibmrepl
Returns all operational attributes that are related to replication.

++ibmpwdpolicy
Returns all operational attributes that are related to password policy.

See *Supported special attributes and associated list of operational attributes table* in *Administering* section in the IBM Security Directory Server documentation. You can know more about the list of specific attributes the server returns for + and ++ attributes.

Note: Server support for + and ++ is optional, and the list of attributes that returned might not include all operational attributes because of security or performance concerns. A server indicates support for the all operational attributes feature by returning the value `1.3.6.1.4.1.4203.1.5.1` in the `supportedfeatures` root DSE attribute.

To know more about the syntax and usage of the command-line utilities, `idsldapsearch` and `ldapsearch`, see *Command Reference* section in the IBM Security Directory Server documentation.

attrsonly
Specifies attribute information. The **`attrsonly`** parameter must be set to 1 to request attribute types only or set to 0 to request both attribute types and attribute values.

sizelimit
Specifies the maximum number of entries to return. The server can set a lower limit which is enforced at the server.

timeout
The `ldap_search_st()` API specifies the local search timeout value. The `ldap_search_ext()` and `ldap_search_ext_s()` APIs specify both the local search timeout value and the operation time limit that is sent to the server within the search request.

The local search timeout relates to the timeout parameter address that is passed to the API, such as `ldap_search_st` and `ldap_search_ext`. The local search timeout structure has two member variables, long int `tv_sec` and long int `tv_usec`.

- long int tv_sec - Represents elapsed time in seconds.

- long int tv_usec - Represents the rest of elapsed time in microseconds.

  Since the timeout value for local search timeout is `tv_sec + tv_usec`, if `tv_sec` is `0` then the timeout value is in microseconds.

  The operation timeout limit relates to the value set in the LDAP handle, `ld`, by using calls to `ldap_set_option` (`ld`, `LDAP_OPT_TIMELIMIT`, `value_in_seconds`).

serverctrls
: Specifies a list of LDAP server controls. This parameter can be set to NULL. For more information about server controls, see "LDAP controls" on page 27.

clientctrls
: Specifies a list of LDAP client controls. This parameter can be set to NULL. For more information about client controls, see "LDAP controls" on page 27.

Output parameters

res
: Contains the result of the asynchronous operation that is identified by **`msgid`**, or returned directly from `ldap_search_s()` or `ldap_search_ext_s()`. This result is passed to the LDAP parsing routines. See "LDAP_RESULT" on page 105.

msgidp
: This result parameter is set to the message ID of the request if the `ldap_search_ext()` call succeeds.

Usage

These routines are used to run LDAP search operations.

The `ldap_search_ext()` API 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`**. Use a subsequent call to `ldap_result()` to obtain the results from the search.

Similar to `ldap_search_ext()`, the `ldap_search()` API initiates an asynchronous search operation and returns the message ID of this operation. If an error occurs, `ldap_search()` returns `-1`, setting the session error in the `LD` structure, which can be obtained by using `ldap_get_errno()`. If successful, use a subsequent call to `ldap_result()` to obtain the results from the search.

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 an LDAP error code if the operation was not successful. For more information about possible errors and how to interpret them, see "LDAP_ERROR" on page 45. If any entries are returned from the search, they are contained in the **`res`** parameter. This parameter is opaque to the caller. Entries, attributes, values, and others, must be extracted by calling the result parsing routines. The memory that is allocated for **`res`** must be freed when no longer in use, whether the operation was successful, by calling `ldap_msgfree()`.

The `ldap_search_ext()` and `ldap_search_ext_s()` APIs support LDAP V3 server controls and client controls, and allow varying size and time limits to be easily specified for each search operation. The `ldap_search_st()` API is identical to `ldap_search_s()`, except that it requires an additional parameter that specifies a local timeout for the search.

There are three options in the session handle `ld` which potentially can affect how the search is run. They are:

LDAP_OPT_SIZELIMIT
A limit on the number of entries that are returned from the search. 0 means no limit. The value from the session handle is ignored when you use 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. Zero means no limit.

Note: The value from the session handle is ignored when you use 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 must be handled during the search. The `LDAP_DEREF_SEARCHING` value means that aliases must be dereferenced during the search but not when you locate the base object of the search. The `LDAP_DEREF_FINDING` value means that aliases must be dereferenced when you locate the base object but not during the search.

These options are set and queried by using the `ldap_set_option()` and `ldap_get_option()` APIs.

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 that is set to `LDAP_SCOPE_BASE`, and filter that is set to `"(objectclass=*)"`. The **`attrs`** parameter optionally 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 list entry, scope set to `LDAP_SCOPE_ONELEVEL`, and filter that is set to `"(objectclass=*)"`. The **`attrs`** parameter optionally contains the list of attributes to return for each child entry. If only the distinguished names of child entries are wanted, the **`attrs`** parameter must specify a NULL-terminated array of one-character strings that has the value `dn`.

Errors

`ldap_search_s()`, `ldap_search_ext_s`, and `ldap_search_st()` return the LDAP error code from the search operation.

`ldap_search()` and `ldap_search_ext()` return `-1` instead of a valid **`msgid`** if an error occurs, setting the session error in the `LD` structure. The session error can be obtained by using `ldap_get_errno()`.

For more information, see “LDAP_ERROR” on page 45.

Notes

These routines allocate storage that is returned by the **res** parameter. Use `ldap_msgfree()` to free this storage.

See also

ldap_result, ldap_error, ldap_sort, ldap_paged_results

LDAP_SERVER_INFORMATION IN DNS

Use the `LDAP_SERVER_INFORMATION` IN DNS API to run various LDAP operations for LDAP server information.

```
ldap_server_locate
ldap_server_free_list
ldap_server_conf_save
```

These LDAP APIs are provided to run the following operations:

- Use LDAP server information that is published in the Domain Name System (DNS) to locate one or more LDAP servers, and associated information. Server information is returned as a linked list of server information structures.
- Free all storage that is associated with a linked list of server information structures.
- Store information about one or more LDAP servers in a local configuration repository. The local configuration can be used to mimic information that can also be published in DNS.

Synopsis

```
#include ldap.h


int ldap_server_locate (
       LDAPServerRequest *server_request,
       LDAPServerInfo    **server_info_listpp);

int  ldap_server_free_list(
        LDAPServerInfo *server_info_listp);

int ldap_server_conf_save(
       char           *filename,
       unsigned long   ttl,
       LDAPServerInfo  *server_info_listp));

typedef struct LDAP_Server_Request {
    int     search_source;      /* Source for server info     */
#define LDAP_LSI_CONF_DNS  0    /* Config first, then DNS (def)*/
#define LDAP_LSI_CONF_ONLY 1    /* Local Config file only     */
#define LDAP_LSI_DNS_ONLY  2    /* DNS only                   */
    char    *conf_filename      /* pathname of config file    */
    int     reserved;           /* Reserved, set to zero      */
    char    *service_key;       /* Service string             */
    char    *enetwork_domain;   /* eNetwork domain (eDomain)  */
    char    **name_servers;     /* Array of name server addrs */
    char    **dns_domains;      /* Array of DNS domains       */
    int     connection_type;    /* Connection type            */
#define LDAP_LSI_UDP_TCP 0      /* Use UDP, then TCP (default)*/
#define LDAP_LSI_UDP 1          /* Use UDP only               */
#define LDAP_LSI_TCP 2          /* Use TCP only               */
    int     connection_timeout; /* connect timeout (seconds)  */
    char    *DN_filter;         /* DN suffix filter           */
    char    *proto_key          /* Symbolic protocol name     */
    unsigned char reserved2[60]; /* reserved fields, set to 0 */
} LDAPServerRequest;


typedef struct LDAP_Server_Info {
    char    *lsi_host;        /* LDAP server's hostname */
    unsigned short lsi_port;  /* LDAP port              */
    char    *lsi_suffix;      /* Server's LDAP suffix   */
    char    *lsi_query_key;   /* service_key[.edomain]  */
    char    *lsi_dns_domain;  /* Publishing DNS domain  */
    int      lsi_replica_type;/* master or replica      */
#define LDAP_LSI_MASTER  1    /* LDAP Master            */
#define LDAP_LSI_REPLICA 2    /* LDAP Replica           */
    int      lsi_sec_type;    /* SSL or non-SSL         */
#define LDAP_LSI_NOSSL   1    /* Non-SSL                */
#define LDAP_LSI_SSL     2    /* Secure Server          */
    unsigned short lsi_priority; /* Server priority     */
    unsigned short lsi_weight; /* load balancing weight */
    char    *lsi_vendor_info;  /* vendor information    */
    char    *lsi_info;        /* LDAP Info string       */
    struct LDAP_Server_Info *prev; /* linked list previous ptr */
    struct LDAP_Server_Info *next; /* linked list next ptr     */
} LDAPServerInfo;
```

Input parameters

server_request

Specifies a pointer to an LDAPServerRequest structure, which must be initialized to zero before you set specific parameters. This setting ensures that defaults are used when a parameter is not explicitly set. If the default behavior is wanted for all possible input parameters, set **server_request** to NULL. This setting is equivalent to setting the LDAPServerRequest structure to zero. Otherwise, supply the address of the LDAPServerRequest structure, containing the following fields:

search_source

Specifies where to find the server information. **search_source** can be one of the following information:

- Access the local LDAP DNS configuration file. If the file is not found, or the file does not contain information for a combination of the **service_key**, **enetwork_domain**, and any of the DNS domains as specified by the application, then access DNS.
- Search the local LDAP DNS configuration file only.
- Search DNS only.

conf_filename

Specifies an alternative configuration file name. Specify NULL to get the default file name and location.

reserved

Represents a reserved area for future function, which must be initialized to zero.

service_key

Specifies the search key. For example, use the service name string when you obtain a list of Service records (SRV), pseudo-SRV Text records (TXT), or CNAME alias records from DNS. If not specified, the default is “ldap”.

Note: Standards are moving towards the use of an underscore (_) as a prefix for service name strings. Over time, it is expected that "_ldap" is the preferred service name string for publishing LDAP services in DNS. If the application does not specify **`service_key`**, and no entries are returned by using the default "ldap." service name, the search is automatically rerun by using "_ldap" as the service name. As an alternative, the application can explicitly specify "_ldap" as the service name. The search is directed specifically at DNS SRV records that use "_ldap" as the service name.

enetwork_domain
Indicates that LDAP servers grouped within the specified eNetwork domain are to be located. An eNetwork domain is a naming construct. It is implemented by the LDAP administrator, to further subdivide a set of LDAP servers (as published in DNS) into logical groupings. By specifying an eNetwork domain, only the LDAP servers that are grouped within the specified eNetwork domain are returned by the `ldap_server_locate()` API. This grouping can be useful when applications require access to a particular set of LDAP servers. For example, the research division within a company might use a dedicated set of LDAP directories, for example, masters and replicas. Applications that require access to information published in research LDAP servers can selectively obtain the host names and ports of research LDAP servers. You can obtain them by publishing this set of LDAP servers in DNS with an eNetwork domain of research. Other LDAP servers that are also published in DNS are not returned.

The criterion for searching DNS to locate the appropriate LDAP servers is constructed by concatenating the following information:

- **`service_key`** (defaults to "ldap")
- **`enetwork_domain`**
- tcp
- DNS domain

For example, if:

- The default **`service_key`** of "ldap" is used
- The eNetwork domain is `sales5`
- The default DNS domain of the client is `midwest.acme.com`

Then, the DNS value that is used to search DNS for the set of LDAP servers that belong to the `sales5` eNetwork domain is `ldap.sales5.tcp.midwest.acme.com`.

If **`enetwork_domain`** is set to zero, the following steps are taken to determine the **`enetwork_domain`**:

- The locally configured default, if set, is used.
- If a locally configured default is not set, then a platform-specific value is used. On a Windows NT operating system, the user logon domain is used.
- If a platform-specific eNetwork domain is not defined, then the eNetwork domain component in the DNS value is omitted. In the preceding example, this results in the following string that is used: `ldap.midwest.tcp.acme.com`.

If **enetwork_domain** is set to a NULL string, then the eNetwork domain component in the DNS value is omitted. This setting might be useful for finding a default eNetwork domain when a specific eNetwork domain is not known.

Note: If the search is run with a non-NULL value for **enetwork_domain**, and the search fails, the search is issued again with a NULL **enetwork_domain**, by using the specified **service_key**, which defaults to `ldap`. The second search with NULL **enetwork_domain** is attempted after a complete search is concluded without results. For example, if **search_source** is set to the default `LDAP_LSI_CONF_DNS`, then the first search is not considered to be complete until both the local configuration and DNS are queried. If both of these searches fail, then both the local configuration and DNS are queried again with a NULL **enetwork_domain**. The intent is to find a set of LDAP servers that are published under the default service key, that is, `ldap`, when nothing can be found published under `ldap.enetwork_domain`. The application can determine whether the located servers are published in an **enetwork_domain** by examining the **lsi_query_key** field, as returned in the `server_info_list` structures that are returned on the `ldap_server_locate()` API. If the returned **lsi_query_key** consists solely of the specified **service_key**, then the located servers were not published in DNS with the specified **enetwork_domain**.
.

name_servers
Specifies a NULL-terminated array of DNS name server IP address in dotted decimal format. For example, 122.122.33.49. If not specified, the locally configured DNS name servers are used.

dns_domains
Specifies a NULL-terminated array of one or more DNS domain names. If not specified, the local DNS domain configuration is used.

Note: The domain names supplied here can take the following forms:
- `austin.ibm.com` (standard DNS format)
- `cn=fred, ou=accounting, dc=austin, dc=ibm, dc=com`

Regarding providing a domain name, these specifications are equivalent. Both result in a domain name of `austin.ibm.com`. This approach makes it easier for an application to locate LDAP servers for binding based on a user name space. This space is mapped into the DNS name space. For more information, see the section *DNS domains and configuration file*.

connection_type
Specifies the type of connection to use when it communicates with the DNS name server. The following options are supported:
- Use UDP first. If no response is received, or data truncation occurs, then use TCP.
- Use only UDP.
- Use only TCP.

If set to zero, the default is to use UDP first (then TCP).

UDP is the preferred connection type, and typically runs well. You might want to consider by using TCP/IP if:

- The amount of data that is returned does not fit in the 512-byte UDP packet.
- The transmission and receipt of UDP packets turns out to be unreliable. This action might depend on network characteristics.

connection_timeout
Specifies a timeout value when querying DNS (for both TCP and UDP). If `LDAP_LSI_UDP_TCP` is specified for **`connection_type`** and a response is not received in the specified time period for UDP, TCP is attempted. A value of zero results in an innumerable timeout. When the `LDAPServerRequest` parameter is set to NULL, the default is 10 seconds. When you pass the `LDAPServerRequest` parameter, this parameter must be set to a nonzero value if an indefinite timeout is not wanted.

DN_filter
Specifies a Distinguished Name to be used as a filter, for selecting candidate LDAP servers that are based on the server suffixes. If the most significant portion of the DN is an exact match with a server suffix (after it normalizes), an `LDAPServerInfo` structure is returned for the server or suffix combination. If it does not match, an `LDAPServerInfo` structure is not returned for the server or suffix combination.

proto_key
Specifies the protocol key. For example, `tcp` or `_tcp`, to be used when you obtain a list of SRV, pseudo-SRV TXT, or CNAME alias records from DNS. If not specified, the default is `tcp`.

Note: Standards are moving towards the use of an underscore (_) as a prefix for the protocol. Over time, it is expected that `_tcp` will become the preferred protocol string for publishing LDAP and other services in DNS. If the application does not specify **`protocol_key`** and no entries are returned by using the default `tcp` protocol key, the search is automatically rerun by using the `_tcp` protocol. As an alternative, the application explicitly specifies `_tcp` as the protocol, and the search is directed specifically at DNS SRV records that use the `_tcp` protocol.

reserved2
Represents a reserved area for future function, which must be initialized to zero.

server_info_listpp
Specifies the address that is set to point to a linked list of `LDAPServerInfo` structures. Each `LDAPServerInfo` structure that is defined in the list contains server information that is obtained from either of the following items:
- DNS
- Local configuration

filename
Specifies an alternative configuration file name. Specify NULL to get the default file name and location.

ttl Specifies the time-to-live, in minutes, for server information that is saved in the configuration file. Set `ttl` to zero if it is intended to be a permanent repository of information.

When the `ldap_server_locate()` API accesses the configuration file with **`search_source`** set to `LDAP_LSI_CONF_ONLY`, and the configuration file is not refreshed in `ttl` minutes, the `LDAP_TIMEOUT` error code is returned.

When the `ldap_server_locate()` API is accesses the configuration file with **`search_source`** set to `LDAP_LSI_CONF_DNS`, and the configuration file is not refreshed in `ttl` minutes, then network DNS is accessed to obtain server information.

server_info_listp
Specifies the address of a linked list of `LDAPServerInfo` structures. This linked list might be returned from the `ldap_server_locate()` API, or might be constructed by the application.

Output parameters

Returns `0` if successful. If an error is encountered, an appropriate return code as defined in the `ldap.h` file is returned. If successful, the address of a linked list of `LDAPServerInfo` structures is returned.

server_info_listpp
Upon successful return from `ldap_server_locate()`, `server_info_listpp` points to a linked list of `LDAPServerInfo` structures. The `LDAPServerInfo` structure contains the following fields:

lsi_host
Fully qualified host name of the target server (NULL-terminated string).

lsi_port
Integer representation of the LDAP server port.

lsi_suffix
String that specifies a supported suffix for the LDAP server (NULL-terminated string).

lsi_query_key
Specifies the eNetwork domain to which the LDAP server belongs, prefixed by the service key. For example, if service key is `ldap` and eNetwork domain is sales, then `lsi_query_key` is set to `ldap.sales`. If the server is not associated with an eNetwork domain (as published in DNS), then `lsi_query_key` consists solely of the service key value. Also, for example, if the service key is `_ldap` and the eNetwork domain is marketing, then `lsi_query_key` is set to `_ldap.marketing`.

lsi_dns_domain
DNS domain in which the LDAP server was published. For example, the DNS search might be for `ldap.sales.tcp.austin.ibm.com`, but the resulting servers have a fully qualified DNS host name of `ldap2.raleigh.ibm.com`. In this example, `lsi_host` is set to `ldap2.raleigh.ibm.com` while `lsi_dns_domain` is set to `austin.ibm.com`. The actual domain in which the server was published might be of interest, particularly when multiple DNS domains are configured or supplied as input.

lsi_replica_type
Specifies the type of server, `LDAP_LSI_MASTER` or `LDAP_LSI_REPLICA`. If set to zero, the type is unknown.

lsi_sec_type
Specifies the port security type, `LDAP_LSI_NOSSL` or `LDAP_LSI_SSL`. This value is derived from the `ldap` or `ldaps` prefix in the LDAP

URL. If the LDAP URL is not defined, the security type is unknown and `lsi_sectype` is set to zero.

lsi_priority
The priority value that is obtained from the SRV RR (or the pseudo-SRV TXT RR). Set to zero if unknown or not available.

lsi_weight
The weight value that is obtained from the SRV RR or the pseudo-SRV TXT RR. Set to zero if unknown or not available.

lsi_vendor_info
NULL-terminated string that is obtained from the `ldapvendor` TXT RR, if defined. It might be used to identify the LDAP server vendor or version information.

lsi_info
NULL-terminated information string that is obtained from the `ldapinfo` TXT RR, if defined. If not defined, `lsi_info` is set to NULL. This information string can be used by the LDAP or network administrator to publish more information about the target LDAP server.

prev Points to the previous **`LDAP_Server_Info`** element in the linked list. This value is NULL if at the top of the list.

next Points to the next **`LDAP_Server_Info`** element in the linked list. This value is NULL if at the end of the list.

Usage

DNS domains and configuration file
The local configuration file can contain server information for combinations of the following information:

- Service key (typically set to `ldap` or `_ldap`)
- eNetwork domain
- DNS domains

When the application sets `search_source` to the default `LDAP_LSI_CONFIG_DNS`, the `ldap_server_locate()` API attempts to find server information in the configuration file for the designated service key, eNetwork domain, and DNS domains.

If the configuration file does not contain information that matches this criteria, the locator API searches the DNS. It searches by using the specified service key, eNetwork domain, and DNS domains. For example:

- The application supplies the following three DNS domains:
  - `austin.ibm.com`
  - `raleigh.ibm.com`
  - `miami.ibm.com`

  Also, the application uses the default service key, that is, `ldap`, and specifies sales for the eNetwork domain.
- The configuration file contains server information for `austin.ibm.com` and `miami.ibm.com`, with the default service key and eNetwork domain of sales.
- Information is also published in DNS for `raleigh.ibm.com`, with the default service key and eNetwork domain of sales.

- The **search_source** parameter is set to LDAP_LSI_CONFIG_DNS, which indicates that both the configuration file and DNS are to be used if necessary.
- The locator API builds a single ordered list of server entries, with the following entries:
  - Server entries for the austin.ibm.com DNS domain, as extracted from the configuration file.
  - Server entries for the raleigh.ibm.com DNS domain, as obtained from DNS over the network.
  - Server entries for the miami.ibm.com DNS domain, as extracted from the configuration file.

The resulting list of servers contains all the austin.ibm.com servers first, followed by the raleigh.ibm.com servers, followed by the miami.ibm.com servers. Within each group of servers, the entries are sorted by priority and weight.

API usage

These routines are used to run operations that are related to finding and saving LDAP server information.

ldap_server_locate()

The ldap_server_locate() API is used to locate one or more suitable LDAP servers. In general, an application uses the ldap_server_locate() API as follows:

- Before you connect to an LDAP server in the enterprise, use ldap_server_locate() to obtain a list of one or more LDAP servers. The servers are published in DNS or in the local configuration file. Typically, an application can use the default request settings by passing a NULL for the LDAPServerRequest parameter. By default, the API looks for server information in the local configuration file first. Then, it moves on to DNS if the local configuration file does not exist or expired.

  Note: If no server entries are found, and the application does not specify the service key, which defaults to ldap, then the ldap_server_locate() function runs the complete search again, by using the alternative "_ldap" for the service key. The results of this second search, if any, are returned to the application.
- After the application obtains the list of servers, it must walk the list, by using the first server that meets its requirements. This action maximizes the advantage that can be derived from using the priority and weighting scheme that is implemented by the administrator. The application might not want to use the first server in the list for several reasons:
  - The client requires to specifically connect by using SSL or non-SSL. For each server in the list, the application can query the rootDSE to determine whether the server supports a secure SSL port. This query is the preferred approach. Alternatively, the application can walk the list until it finds a server entry with the appropriate security type. An LDAP server might be listening on both an SSL and non-SSL port. In this case, the server has two entries in the server list:
  - The client specifically requires to connect to a Master or Replica.

- The client requires to connect to a server that supports a particular suffix.

  Note: Specify `DN_filter` to filter out servers that do not have a suffix. The DN is under this suffix. To confirm that a server actually supports the suffix, query the server `rootDSE`.
- Some other characteristic that is associated with the wanted server exists, defined in the `ldapinfo` string.

- After the client selects a server, it then issues the `ldap_init` or `ldap_ssl_init` API. If the selected server is unavailable, the application is free to move down the list of servers. It moves the server list until either it finds a suitable server it can connect to, or the list is exhausted.

ldap_server_free_list()

To free the list of servers and associated `LDAPServerInfo` structures, the application must use the `ldap_server_free_list()` API. The `ldap_server_free_list()` API frees the linked list of `LDAPServerInfo` structures and all associated storage as returned from the `ldap_server_locate()` API.

ldap_server_conf_save()

The `ldap_server_conf_save()` API is used to store server information into local configuration. The format for specifying the server information about the `ldap_server_conf_save()` API is identical to the format returned from the `ldap_server_locate()` API.

The application that writes information into the configuration file can specify an optional `time-to-live` for the information that is stored in the file. When an application uses the locator API to access DNS server information, the configuration file is considered to be stale if:

```
date/time_file_last_updated + ttl > current_date/time
```

If the application uses the default behavior for using the configuration file, it bypasses a stale configuration file. It attempts to find all required information from DNS. Otherwise, the `ttl` must be set to zero (indefinite `ttl`), in which case the information is considered to be good indefinitely.

Setting a nonzero `ttl` is most useful when an application or other mechanism exists for refreshing the local configuration file on a periodic basis.

Note: Subsecond response time can be expected in many cases, when you use UDP to query DNS. Since most applications get the server information during initialization, repetitive invocation of the locator API is usually unnecessary.

By default, the configuration file is stored in the following platform-specific location:

UNIX `/etc/ldap_server_info.conf`

Windows NT and Windows 2000
`\drivers\etc\ldap_server_info.conf`

Format of local configuration file

The following sample shows the definition for a local configuration file that is created with the `ldap_server_conf_save()` API. You must create the file by using the `ldap_server_conf_save()` API. However, with careful editing, it can also be created and maintained manually. Some basic rules for managing this file manually:

- Comment fields must begin with a number sign (#). Comment fields are ignored.
- All parameters are positional.
- The first non-comment line must contain the time-to-live value for the file.

```
################################################################
# Local LDAP DNS configuration file.
#
# The following line holds the file's expiration time, which is
# a UNIX time_t value (time in seconds since January 1, 1970 UTC).
# A value of 0 indicates that the file will not expire.
#907979782
0
# Each of the following lines in this file represents a known
# LDAP server. The lines have the following format:
#
# service domain host priority weight port replica sec "suffix"
        "vendor info" "general info"
#
# where:
#
#  service= service_key[.eNetwork_domain]
#
#  domain=  DNS domain
#
#  host=    fully qualified DNS name of the LDAP Server host
#
#  priority= target host with the lowest priority tried first
#
#  weight=  load balancing method.  When multiple hosts have the
#           same priority, the host to be contacted first is
#           determined by the weight value.
#           Set to 0 if load balancing is not needed.
#
#  port=    The port to use to contact the LDAP Server.
#
#  replica= Use "1" to indicate Master.
#           "2" to indicate Replica.
#
#  sec=     Use "1" to indicate Non-SSL
#           "2" to indicate SSL.
#
#  suffix=  A suffix on the server.
#
#  vendor info= a string that identifies the LDAP server vendor
#
#  general info=    Any informational text you wish to include.
#
ldap     austin.ibm.com ldapserver1.austin.ibm.com 1 1 389 1 1
        "ou=users,o=sample" "IBM SecureWay" "phoneinfo"
ldap     austin.ibm.com ldapserver2.austin.ibm.com 1 1 389 2 1
        "ou=users,o=sample" "IBM SecureWay" "phoneinfo replica"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
        "cn=GSO,o=sample"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
        "ou=Austin,o=sample" "IBM" "GSO ePersonbase"
```

```
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
        "cn=GSO,o=sample"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
        "ou=Austin,o=sample" "IBM" "GSO ePersonbase"
ldap.sales raleigh.ibm.com saleshost1.raleigh.ibm.com 1 1 389 1 1
        "dc=raleigh,dc=ibm, dc=com" "IBM" "Sales Marketing"
ldap.sales raleigh.ibm.com saleshost2.raleigh.ibm.com 2 1 389 2 1
        "dc=raleigh,dc=ibm, dc=com" "IBM" "Sales Marketing Replica"
#
##################################################################
```

The newer form of service keys can also be used in the configuration file. For example, the following code is an excerpt that uses _ldap as the service key:

```
_ldap     austin.ibm.com ldapserver1.austin.ibm.com 1 1 389 1 1
        "ou=users,o=sample" "IBM SecureWay" "phoneinfo"
_ldap     austin.ibm.com ldapserver2.austin.ibm.com 1 1 389 2 1
        "ou=users,o=sample" "IBM SecureWay" "phoneinfo replica"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "" ""
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
        "cn=GSO,o=sample"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
        "ou=Austin,o=sample" "IBM" "GSO ePersonbase"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "" ""
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
        "cn=GSO,o=sample"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
        "ou=Austin,o=sample" "IBM" "GSO ePersonbase"
_ldap.sales raleigh.ibm.com saleshost1.raleigh.ibm.com 1 1 389 1 1
        "dc=raleigh,dc=ibm,dc=com" "IBM" "Sales Marketing"
_ldap.sales raleigh.ibm.com saleshost2.raleigh.ibm.com 2 1 389 2 1
        "dc=raleigh,dc=ibm,dc=com" "IBM" "Sales Marketing Replica"
```

Publishing LDAP server information in DNS

If DNS is used to publish LDAP server information, the LDAP administrator must configure the relevant DNS name servers with the appropriate SRV and TXT records. The records reflect the LDAP servers available in the enterprise.

- If SRV records are supported by the DNS servers in the enterprise, SRV records can be created that identify the LDAP servers. They identify with appropriate weighting and priority settings. For more information about SRV records and how they are used, see A. Gulbrandsen, P. Vixie, "A DNS RR for Specifying the Location of Services (DNS SRV)", Internet RFC 2782, Troll Technologies, Vixie Enterprises, February, 2000, which obsoletes RFC 2052.
- TXT records must be associated with the A record of each LDAP server. The TXT records include the LDAP URL records which specify host name, port, base DN, and port type. For example, `ldap` for non-SSL, and `ldaps` for SSL.
- If SRV records are not used, the list of available servers must be specified with a set of TXT records which emulate the SRV RR format.

The LDAP server locator API:

- Provides access to a list of LDAP servers. By default, the locator API queries a local configuration file for the required information. If the file was updated with a nonzero `time-to-live`, and the file is stale, or the file does not contain the required information, the locator API then accesses DNS. By default, the local configuration file has no `time-to-live`, and is considered to be good indefinitely.

Note: The configuration file is designed to hold the same level of information per server that can be obtained from DNS.

- Gathers data relevant to each of the LDAP servers from DNS, by using three sequenced algorithms:
  1. SRV records
  2. Pseudo-SRV records (by using TXT records)
  3. A CNAME alias that references a single host A record

  The algorithms are attempted in sequence until results are returned for one of the algorithms. For example, if no SRV records are found, but pseudo-SRV records are found, the list of servers is built from the pseudo-SRV records.
- Builds a list of LDAP servers, with the first server in the list that is classified as the preferred or default server. Depending on how DNS publishes LDAP servers, the preferred LDAP server can be a reflection of how the administrator organizes the LDAP information in DNS. The application has access to the additional data that was retrieved from DNS. The additional information for each LDAP server information structure can consist of the following information:
  - Host name and port
  - eNetwork domain of the server
  - Fully qualified DNS domain where the host name is published
  - Suffix
  - Replication type (master or replica)
  - Security type (SSL or non-SSL)
  - Vendor ID
  - Administrator-defined data

The application can use `ldap_server_locate()` to obtain a list of one or more LDAP servers that exist in the enterprise. It is published in either DNS or the local configuration file. The additional data might be used by the application to select the appropriate server. For example, the application might require a server that supports a specific suffix, or might require to specifically access the master for update operations.

As input to the API, the application can supply:

- A list of one or more DNS name server IP addresses. The default is to use the locally configured list of name server addresses. When an active name server is located, it is used for all subsequent processing.
- The service key. The default is `ldap`. The service key is used to query DNS for information specific to the LDAP protocol. For example, when you search for SRV records in the `austin.ibm.com` DNS domain, the search is for `ldap.tcp.austin.ibm.com` with `type=SRV`. This example assumes that the search does not include an eNetwork domain component. The application can also specify `_ldap` as the service key and `_tcp` for the protocol, in which case the search is for `_ldap._tcp.austin.ibm.com` with `type=SRV`.
- The name of the eNetwork domain. The eNetwork domain is typically the name that is used to identify the LDAP user authentication domain. It further qualifies the search for relevant LDAP servers, as published in the user DNS domain. For example, when you search for SRV records in

the `austin.ibm.com` DNS domain, with an eNetwork domain of marketing the search is for `ldap.marketing.tcp.austin.ibm.com` with `type=SRV`.

- A list of one or more fully qualified DNS domain names. The default is to use the locally configured domains.

  If multiple domains are supplied, either in the default configuration or explicitly supplied by the application, information is gathered from each DNS domain. The server information that is returned from the locator API is grouped by DNS domain. If two domains are supplied, for example, `austin.ibm.com` and `raleigh.ibm.com`, the entries for LDAP servers that are published in the `austin.ibm.com` domain is first in the list, with the `austin.ibm.com` servers that are sorted by priority and weight. Entries for LDAP servers that are published in the `raleigh.ibm.com` domain follows the entire set of `austin.ibm.com` servers (with the `raleigh.ibm.com` servers that are sorted by priority and weight).

  Note: All entries that are returned by the locator API are associated with a single *service_key.edomain* combination.

  DNS domain names that are supplied here can take two forms:

  – `austin.ibm.com` (standard DNS format)

  – `cn=fred, ou=accounting, dc=austin, dc=ibm, dc=com`

  About providing a fully qualified DNS domain name, these provisions are equivalent. Both result in a DNS domain name of `austin.ibm.com`. This approach makes it easier for an application to locate LDAP servers it requires to bind with, based on a user name space. This space is mapped into the DNS name space.
- The connection type (UDP or TCP).
- A DN for comparison against the suffix that are defined for each LDAP server entry. This string, if supplied, is used as a filter. Only server entries that define a suffix that compares with the DN are returned by the locator API. For example, a DN of `"cn=fred, ou=accounting, o=sample"` matches the first of the following DN, but not the second:

  – `o=sample`

  – `o=tivoli, c=us`

  The ability to filter based upon each LDAP server suffix is supplied as a convenience. Therefore, the application does not require to step through the list of servers, comparing a DN with each entry suffix.
- The application can specify how information in the local configuration file is used. The default is to look in the local, configuration file for the wanted information. If the information is not found, then DNS servers on the network are accessed. The application can specify the following information:

  – Look in the configuration file first, then access the network (default).

  – Look in the configuration file only.

  – Access DNS only.

  When you use the default configuration file, the application is not required to specify the location. Alternatively, the application can provide a path name to a configuration file.

Note: Information that is stored in the configuration file takes the same form as information obtained from DNS. The difference is that it is saved in the file by an application. The file can also be constructed and distributed to users by the administrator.
Maximum benefit is obtained when applications can use the defaults for all the parameters. This benefit minimizes application knowledge of the specifics that is related to locating LDAP servers.

Using SRV and TXT records

The DNS-lookup routine looks for SRV records first. If one or more servers are found, then the server information is returned. The second algorithm, which is based on TXT records that emulate SRV records, is not called.

Use the SRV records for finding the address of servers and for a specific protocol and domain. This use is described in *RFC 2052, "A DNS RR for Specifying the Location of Services (DNS SRV)"*. Correct use of the SRV RR grants the administrator the following actions:

- Distribute a service across multiple hosts within a domain
- Move the service from host to host without disruption
- Designate certain hosts as primary and others as alternates, or backups, by using a priority and weighting scheme

TXT stands for text. TXT records are strings. BIND versions before 4.8.3 do not support TXT records. To fully implement the technique that is described in RFC 2052, the DNS name servers must use a version of BIND. This version supports SRV records and TXT records. An SRV resource record (RR) has the following components, as described in RFC 2052:

```
service.proto.name ttl class SRV priority weight port target
```

where:

service
Symbolic name of the wanted service. By default, the service name or service key is `ldap`. When used to publish servers that are associated with an eNetwork domain, the service value is derived by concatenating the service key, for example, `ldap`. With the eNetwork domain name, for example, marketing. In this example, the resulting service is `ldap.marketing`.

proto Protocol, typically `tcp` or `udp`, or `_tcp` or `_udp`.

name Domain name that is associated with the RR.

ttl Time-to-live, standard DNS meaning.

class Standard DNS meaning. For example, IN.

Priority
Target host with lowest number priority must be attempted first.

weight
Load balancing mechanism. When multiple target hosts have the same priority, the chance of contacting one of the hosts first must be proportional to its weight. Set to `0` if load balancing is not necessary.

port Port on the target host for the service.

target Target host name must have one or more A records that are associated with it.

The approach is to use SRV records to define a list of candidate LDAP servers. Then, use TXT records that are associated with the A record of each host to get more information about each LDAP server. Three forms of TXT records are understood by the LDAP client DNS lookup routines:

- The service TXT record provides a standard LDAP URL, that is, provides host, port and base DN.
- The ldaptype TXT record identifies whether the LDAP server is a master or replica.
- The ldapvendor TXT record identifies the vendor.

```
ldap         A      199.23.45.296
             TXT    "service:ldap://ldap.ibm.com:389/o=foo,c=us"
             TXT    "ldaptype: master"
             TXT    "ldapvendor: IBMeNetwork"
             TXT    "ldapinfo: ldapver=3, keyx=fastserver"
```

The ldapinfo freeform TXT record provides more information, as defined by the LDAP or network administrator. As in the example above, the information can be keyword-based. The ldapinfo record is available to the application.

In combination, the name server might contain the following information, which effectively publishes the set of LDAP servers that are in the marketing eNetwork domain:

```
ldap.marketing.tcp    SRV    0  0  0    ldapm
                      SRV    0  0  0    ldapmsec
                      SRV    0  0  0    ldapmsuffix
                      SRV    1  1  0    ldapr1
                      SRV    1  2  0    ldapr2
                      SRV    1  2  0    ldapr2sec
                      SRV    2  1  2222 ldapr3.raleigh.ibm.com.

ldapm        A      199.23.45.296
             TXT    "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
             TXT    "ldaptype: master"

ldapmsec     A      199.23.45.296
             TXT    "service:ldaps://ldapm.austin.ibm.com:686/o=foo,c=us"
             TXT    "ldaptype: master"

ldapmsuffix  A      199.23.45.296
             TXT    "service:ldaps://ldapm.austin.ibm.com:389/o=moo,c=us"
             TXT    "ldaptype: master"

ldapr1       A      199.23.45.297
             TXT    "service:ldap://ldapr1:389/o=foo,c=us"
             TXT    "ldaptype: replica"

ldapr2       A      199.23.45.298
             TXT    "service:ldap://ldapr2:389/o=foo,c=us"
             TXT    "ldaptype: replica"

ldapr2sec    A      199.23.45.298
             TXT    "service:ldaps://ldapr2/o=foo,c=us"
             TXT    "ldaptype: replica"
             TXT    "ldapinfo: ca=verisign, authtype=server"

ldapr3.raleigh.ibm.com.   A   199.23.45.299
```

In this example, a DNS search for ibmldap.marketing.tcp.austin.ibm.com with type=SRV returns seven SRV records, which represent entries for four hosts. An SRV record is required for each port or suffix combination that is supported by a server. For example, a server that supports an SSL and non-SSL port might have at least two SRV records and two corresponding A records. These records point to the same IP address. In this example, the A RR combinations for ldapm/ldapmsec/ldapmsuffix and ldapr2/ldapr2sec map to the same host address.

Note: ldapmsuffix provides an alternative suffix for the 199.23.45.296 host.

The port that is specified on the SRV record is ignored if the target host has a TXT record that contains an LDAP URL. If the URL is specified without a port, the default port is used (389 for non-SSL, 686 for SSL).

Some rules for constructing strings that are associated with the TXT records:

- If the string contains white space, the entire string that follows TXT must be enclosed in double quotation marks.
- If the string contains characters that are not supported by DNS. For example, the suffix might contain characters that are not supported by DNS, an escape is supported, based on the technique that is described in "Uniform Resource Locators (URL)", Internet RFC 1738, December 1994. For example:

  ```
  TXT     "service:ldaps://ldapr2/o=foo%f0,c=us"
  ```

  grants the x'f0' character to be included in the LDAP URL.

The algorithm for the use of LDAP servers is outlined as follows. The LDAP servers are ordered in the list that is based on this algorithm. The application has the freedom of using the first server in the list that is based on priority and weight. It also has the freedom to select a different server, which is based upon its requirements.

Using pseudo-SRV TXT records

If the SRV algorithm does not return any servers, the secondary algorithm is called. Instead of looking for SRV records, the lookup routine runs a TXT query. It uses the service name string that is supplied on ldap_server_locate(), which defaults to ldap.tcp.

The intent is to emulate the scheme that is provided with SRV records, but by using a search for TXT records instead. To duplicate the previous example by using TXT records instead of SRV records, the following definition is used:

```
ldap.marketing.tcp    TXT     0  0  0    ldapm
                      TXT     0  0  0    ldapmsec
                      TXT     0  0  0    ldapmsuffix
                      TXT     1  1  0    ldapr1
                      TXT     1  2  0    ldapr2
                      TXT     1  2  0    ldapr2sec
                      TXT     2  1  2222 ldapr3.raleigh.ibm.com.

ldapm         A       199.23.45.296
              TXT     "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
```

```
             TXT     "ldaptype: master"

ldapmsec     A       199.23.45.296
             TXT     "service:ldaps://ldapm.austin.ibm.com:686/o=foo,c=us"
             TXT     "ldaptype: master"

ldapmsuffix  A       199.23.45.296
             TXT     "service:ldaps://ldapm.austin.ibm.com:389/o=moo,c=us"
             TXT     "ldaptype: master"

ldapr1       A       199.23.45.297
             TXT     "service:ldap://ldapr1:389/o=foo,c=us"
             TXT     "ldaptype: replica"

ldapr2       A       199.23.45.298
             TXT     "service:ldap://ldapr2:389/o=foo,c=us"
             TXT     "ldaptype: replica"

ldapr2sec    A       199.23.45.298
             TXT     "service:ldaps://ldapr2/o=foo,c=us"
             TXT     "ldaptype: replica"
             TXT     "ldapinfo: ca=verisign, authtype=server"

ldapr3.raleigh.ibm.com.   A   199.23.45.299
```

The LDAP resolver routine assumes that the default domain is in effect when the SRV-type TXT records do not contain fully qualified domain names.

Note: The pseudo-SRV TXT records, in many cases, can exactly replicate the syntax of SRV records, with the exception that SRV is replaced by TXT. This replication makes for consistent parsing of the records by the resolver routines. It also makes it simple to switch between the two mechanisms when you insert this information into the DNS database. However, some versions of DNS require data that is associated with the TXT records to be enclosed in double quotation marks, as follows:

```
ldap.marketing.tcp    TXT     "0  0  0    ldapm"
                      TXT     "0  0  0    ldapmsec"
```

The `ldap_server_locate()` API handles either format.

Using a CNAME alias record

If the pseudo-SRV algorithm does not return any servers, the third algorithm is called. Instead of looking for TXT records, the lookup routine runs a standard query by using the service name string that is supplied on `ldap_server_locate()`, which defaults to `ldap`.

```
ldap.marketing.tcp    CNAME   ldapm

ldapm        A       199.23.45.296
             TXT     "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
             TXT     "ldaptype: master"
```

If TXT records are not associated with the A record, defaults are assumed for port and `ldaptype`.

Alternative scheme for publishing LDAP server information in DNS

A more recent Internet Engineering Task Force (IETF) draft describes a scheme where service keys and the protocol are prefixed with an underscore (_). For more information about this new scheme, see the following internet draft: A. Gulbrandsen, P. Vixie, "A DNS RR for Specifying the Location of Services (DNS SRV)", Internet RFC 2052, Troll Technologies, Vixie Enterprises. January 1999

When services are published in DNS by using the approach that is proposed in this IETF draft, service names and protocol are prefixed with an underscore (_).

For instance, a previous example might be defined as follows:

```
_ldap.marketing._tcp    SRV     0  0  0    ldapm
                        SRV     0  0  0    ldapmsec
                        SRV     0  0  0    ldapmsuffix
                        SRV     1  1  0    ldapr1
                        SRV     1  2  0    ldapr2
                        SRV     1  2  0    ldapr2sec
                        SRV     2  1  2222 ldapr3.raleigh.ibm.com.
```

If all LDAP service information is published within your enterprise this way, the application can choose to not specify service key or protocol, and the `ldap_server_locate()` API first runs its search by using `ldap` and `tcp`. The search does not find any entries, and the API automatically runs the search again by using `_ldap` and `_tcp` for service key and protocol. The search returns the information that is published with the alternative scheme.

If information is published with both schemes, the application must explicitly define the service key and protocol, to ensure that the wanted information is returned.

Errors

`ldap_server_locate()`, `ldap_server_free_list` and `ldap_server_conf_save()` return the LDAP error code that results from the operation.

For more information, see “LDAP_ERROR” on page 45.

See also

ldap_error

LDAP_SSL

Use the `LDAP_SSL` API or LDAP routine for initializing the Secure Socket Layer (SSL) function for an LDAP application, and creating a secure connection to an LDAP server.

`ldap_ssl_client_init`

`ldap_ssl_init`

`ldap_ssl_start` (deprecated)

`ldap_set_cipher`

`ldap_ssl_set_fips_mode_np`

For `ldap_ssl_set_fips_mode_np()`, the FIPS processing mode is set before you create an SSL environment that is used for securing server connections.

Synopsis

```
#include ldap.h
#include ldapssl.h

int ldap_ssl_client_init(
       char      *keyring,
       char      *keyring_pw,
       int       ssl_timeout,
       int       *pSSLReasonCode);

LDAP *ldap_ssl_init(
       char       *host,
       int        port,
       const char  *name);

int ldap_ssl_start(
       LDAP      *ld,
       char      *keyring,
       char      *keyring_pw,
       char      *name);

int ldap_set_cipher(
       LDAP      *ld,
       char      *option);

int ldap_ssl_set_fips_mode_np(
   int        mode);
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()` or `ldap_open()`.

host
Several methods are supported for specifying one or more target LDAP servers, including the following ones:

Explicit host list
Specifies the name of the host the LDAP server runs on. The host parameter can contain a blank-separated list of hosts to connect to, and each host might optionally be of the form **`host`**:**`port`**. If present, the:**`port`** overrides the port parameter that is supplied on `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`. The following examples are typical:

```
ld=ldap_ssl_init ("server1", ldap_port, name);
ld=ldap_ssl_init ("server2:636, ldap_port, name);
ld=ldap_ssl_init ( "server1:636 server2:2000 server3",
        ldap_port, name);
```

Local host
If the host parameter is NULL, the LDAP server is assumed to be running on the local host.

Default hosts
If the host parameter is set to `ldaps://`, the LDAP library attempts to locate one or more default LDAP servers, with secure SSL ports, by using the `ldap_server_locate()` function. The port that is specified on the call is ignored because `ldap_server_locate()` returns the port. For example, the following two are equivalent:

```
ld=ldap_ssl_init ("ldaps://", ldap_port, name);
ld=ldap_ssl_init (LDAPS_URL_PREFIX, LDAPS_PORT, name);
```

Note: `ldaps` or `LDAPS_URL_PREFIX` must be used to obtain servers with secure ports. If more than one default server is located, the list is processed in sequence until an active server is found.

The LDAP URL can include a Distinguished Name, which is used as a filter for selecting candidate LDAP servers that are based on the server suffixes. The server is added to the list of candidate servers. This addition is done if the most significant portion of the DN is an exact match with a server suffix after normalizing for case. For example, the following returns default LDAP servers that have a suffix that supports the specified DN only:

```
ld=ldap_ssl_init ("ldaps:///cn=fred, dc=austin, dc=ibm,
       dc=com", LDAPS_PORT, name);
```

In this case, a server that has a suffix of "`dc=austin, dc=ibm, dc=com`" matches. If more than one default server is located, the list is processed in sequence until an active server is found.

If the LDAP URL contains a host name and optional port, the host is used to create the connection. No attempt is made to locate the default servers, and the DN, if present, is ignored. For example, the following two are equivalent:

```
ld=ldap_ssl_init ("ldaps://myserver", LDAPS_PORT, name);
ld=ldap_ssl_init ("myserver", LDAPS_PORT, name);
```

For more information about the algorithm that is used to locate default LDAP servers, see “Locating default LDAP servers” on page 78.

Host with privileged port

On platforms that support the `rresvport` function (typically UNIX platforms), if a specified host is prefixed with "`privport://`", then the LDAP library uses the `rresvport()` function to attempt to obtain one of the reserved ports (512 through 1023), instead of an ephemeral port. The search for a reserved port starts at 1023 and stops at 512. If a reserved port cannot be obtained, the function call fails. For example:

```
 ld=ldap_ssl_init ("privport://server1, ldap_port, name);
ld=ldap_ssl_init ("privport://server2:1200, ldap_port,
       name);
ld=ldap_ssl_init ( "privport://server1:800 server2:2000
       privport://server3", ldap_port, name);  port
```

port Specifies the port number to connect to. If you want the default IANA-assigned SSL port of 636, specify `LDAPS_PORT`.

keyring

Specifies the name of a key database file (with `kdb` extension). The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. A key database can also be used to store the client private keys and associated client certificates. A private key and associated client certificate are required only if the LDAP server is configured to require client and server authentication. If the LDAP server is configured to provide only server authentication, a private key and client certificate are not required.

Default keyring and password

Applications can use the default keyring file, as installed with the LDAP support, by specifying NULL pointers for keyring and **`keyring_pw`**. The default keyring file, that is, `ldapkey.kdb`, and the associated password stash file, that is, `ldapkey.sth`, are installed in the `/etc` directory under *LDAPHOME*, where *LDAPHOME* is the path to the installed LDAP support. *LDAPHOME* varies by operating system platform:

- For AIX, Solaris, and HP-UX (Itanium): `/opt/IBM/ldap/VERSION_NUMBER`
- Linux: `/opt/ibm/ldap/VERSION_NUMBER`
- Windows: `C:\Program Files\IBM\LDAP\VERSION_NUMBER`

  Note: This location is the default installation location. The actual *LDAPHOME* is determined during installation.

where, `VERSION_NUMBER` is `V6.3.1` for IBM Security Directory Server 6.3.1.

Applications typically use the default `keyring` file when the LDAP servers used by the applications are configured with X.509 certificates. These certificates are issued by one of the well-known defaults CA. A trusted root key is the public key and associated Distinguished Name of a CA. The following trusted roots are automatically defined in the default LDAP key database file (`ldapkey.kdb`):

- Integrion Certification Authority Root
- IBM World Registry Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freeemail CA
- Thawte Personal Basic CA
- Thawte Premium Server CA
- VeriSign Test CA Root Certificate
- RSA Secure Server Certification Authority
- VeriSign Class 1 Public Primary Certification Authority
- VeriSign Class 2 Public Primary Certification Authority
- VeriSign Class 3 Public Primary Certification Authority
- VeriSign Class 4 Public Primary Certification Authority

Note: Each of these certificates is initially set to be trusted.
If the default `keyring` file cannot be located, this set of trusted roots is also built-in to the LDAP or SSL code, and is used by default.

By modifying the contents of `ldapkey.kdb`, as in *`LDAPHOME`*`\etc`, all LDAP applications that use SSL and specify NULL pointers to **`keyring`** and **`keyring_pw`** use the revised key database without change to each application. There are various reasons for changing or customizing a `keyring` file, including:

- Adding one or more new trusted roots (that is, adding trust for more CAs).
- Removing trust. For example, your enterprise might obtain all of its server certificates from VeriSign. In this case, it is appropriate to mark the VeriSign certificates as trusted only.

Note: For the default LDAP keyring file to be generally useful to a set of applications, it requires to be readable by each of the applications. It is not suitable to store client certificates with private keys in a `keyring` file that is readable by users other than the owner of the private keys. Therefore, the client certificates with private keys must not be stored in the default LDAP keyring file. They must be stored in keyring files that can be accessed by the appropriate user only. Care must be taken to ensure that local file system permissions are set so that the `keyring` file and associated stash file, if used, are accessible by the appropriate user only. The password that is defined for the default `ldapkey.kdb` file is `ssl_password`. Use this password when initially accessing the default keyring database with the **`ikeyman`** utility. This default password is also encrypted into the default `keyring` password stash file, `ldapkey.sth`, in the same directory as `ldapkey.kdb`. Use the **`ikeyman`** utility to change the password.

If `keyring` is specified, you must specify a file name with fully qualified path. If a file name without a fully qualified path is specified, the LDAP library looks in the current directory for the file. The key database file that is specified here must be created by using the **`ikeyman`** utility.

For more information about using **`ikeyman`** to manage the contents of a key database, see the *Using iKeyman* section in the *Administering* section in the IBM Security Directory Server documentation.

Note: Although still supported, use of the `ldap_ssl_start()` is discouraged, as its use is deprecated. Any application that uses the `ldap_ssl_start()` API must use a single key database per application process only.

keyring_pw
Specifies the password that is used to protect the contents of the key database. This password is important, particularly when it protects one or more private keys that are stored in the key database. The password is specified when the key database is initially created, and can be changed by using the **`ikeyman`** utility. In lieu of specifying the password each time the application opens the keyring database, the password can be obtained from a password stash file. This file contains an encrypted version of the password. The password stash file can be created by using the **`ikeyman`** utility. To obtain the password from the password stash file, specify a `NULL` pointer for `keyring_pw`. It is assumed that the password stash file has the same name as the keyring database file, but with an extension of `.sth` instead of `.kdb`. It is also assumed that the password stash file is in the same directory as the keyring database file.

Note: The default keyring file (`ldapkey.kdb`) is initially configured to have `ssl_password` as its password. This password is also initially configured in the default password stash file (`ldapkey.sth`).

name Specifies the name, or label, which is associated with the client private key or certificate pair in the key database. It is used to uniquely identify a private key or certificate pair, as stored in the key database, and might be something like: Digital ID for Fred Smith.

If the PKCS#11 interface is used to run SSL connection by using a Crypto device, then the user must pass the token label of the Crypto device and the certificate that requires to be used for the connection in the following format: `TOKENLABEL:CERTIFICATENAME`. Here, the certificate is stored in the key storage device by using this format.

If the LDAP server is configured to run Server Authentication, a client certificate is not required and name can be set to `NULL`. If the LDAP server is configured to run Client and Server Authentication, a client certificate is required. name can be set to `NULL` if a default certificate or private key pair is designated as the default. See the *Using iKeyman* section in the *Administering* section in the IBM Security Directory Server documentation. Similarly, name can be set to `NULL` if there is a single certificate or private key pair in the designated key database.

ssl_timeout
Specifies the SSL timeout value in seconds. The timeout value controls the frequency with which the SSL protocol stack regenerates session keys. If `ssl_timeout` is set to `0`, the default value `SSLV3_CLIENT_TIMEOUT` is used. Otherwise, the value that is supplied is used, provided it is less than or equal to 86,400 (number of seconds in a day). If `ssl_timeout` is greater than 86,400, then `LDAP_PARAM_ERROR` is returned.

pSSLReasonCode
Specifies a pointer to the SSL Reason Code, which provides more information in case an error occurs during initialization of the SSL stack, when `ldap_ssl_client_init()` is called. See `ldapssl.h` for reason codes that can be returned.

mode For `ldap_ssl_set_fips_mode_np()`, **`mode`** specifies whether FIPS processing mode must be `on` (`1`) or `off` (`0`).

Usage

The US Government regulations about the export of SDKs which provides support for encryption continue to evolve.

The point of control, regarding available levels of encryption, is now the application.

Any LDAP application that uses IBM Security Directory Server C-Client SDK Version 6.0 or later with the supported GSKit, version 6.0.3 or later provides default access to SSL encryption algorithms.

The `ldap_ssl_client_init()` routine is used to initialize the SSL protocol stack for an application process. Initialization includes establishing access to the specified key database file. The `ldap_ssl_client_init()` API must be called one time per application process before you make any other SSL-related LDAP calls, such as `ldap_ssl_init()`. When `ldap_ssl_client_init()` is successfully called, any subsequent invocations return a return code of `LDAP_SSL_ALREADY_INITIALIZED`. This return also means that a particular key database file is effectively bound to an application process. To change the key database, the application or one of its processes must be restarted.

Note: The `ldap_ssl_client_init()` routine is deprecated but is still supported.

The ldap_ssl_environment_init() routine can be used instead of ldap_ssl_client_init() with the advantage of being able to be called more than one time in the same process. Each call creates an SSL environment which is used for subsequent SSL sessions that are initiated by calling ldap_ssl_init(). These SSL environments persist until the LDAP sessions that were created by using them persist.

The ldap_ssl_init() routine is the SSL equivalent of ldap_init(). It is used to initialize a secure SSL session with a server.

Note: The server is not contacted until an operation is run that requires it, allowing various options to be set after initialization.
After the secure connection is established for the LDAP session, all subsequent LDAP messages that flow over the secure connection are encrypted, including the ldap_simple_bind() parameters, until ldap_unbind() is called.

ldap_ssl_init() returns a session handle, a pointer to an opaque data structure that must be passed to subsequent calls that pertain to the session. These subsequent calls return NULL if the session cannot actually be established with the server. Use ldap_get_option() to determine why the call failed.

The LDAP session handle that is returned by ldap_ssl_init and ldap_init is a pointer to an opaque data type representing an LDAP session. The ldap_get_option() and ldap_set_option() APIs are used to access and set various session-wide parameters. For more information about ldap_get_option() and ldap_set_option(), see "LDAP_INIT" on page 63.

Note: When you connect to an LDAP V2 server, one of the ldap_simple_bind() or ldap_bind() calls must be completed before other operations can be run on the session, except for ldap_set/get_option(). The LDAP V3 protocol does not require a bind operation before you run other operations.

Although still supported, the use of the ldap_ssl_start() API is now deprecated. The ldap_ssl_client_init() and ldap_ssl_init() APIs must be used instead. The ldap_ssl_start() API starts a secure connection to an LDAP server by using SSL. ldap_ssl_start() accepts the ld from an ldap_open() and runs an SSL handshake to a server. ldap_ssl_start() must be called after ldap_open() and before ldap_bind(). When the secure connection is established for the ld, all subsequent LDAP messages that flow over the secure connection are encrypted, including the ldap_bind() parameters, until ldap_unbind() is called.

The following scenario depicts the calling sequence, where the entire set of LDAP transactions is protected by using a secure SSL connection, including the dn and password that flow on the ldap_simple_bind():

```
rc = ldap_ssl_client_init (keyfile, keyfile_pw, timeout,
       &reasoncode);
ld = ldap_ssl_init(ldaphost, ldapport, label );
rc = ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, &ciphers);
rc = ldap_simple_bind_s(ld, binddn, passwd);

...additional LDAP API calls

rc = ldap_unbind( ld );
```

Note: The sequence of calls for the deprecated APIs is ldap_open/init(), ldap_ssl_start(), followed by ldap_bind().

The following ciphers are attempted for the SSL handshake by default, in the order shown:

```
AES_256
AES_128
RC4_SHA_US
RC4_MD5_US
DES_SHA_US
3DES_SHA_US
RC4_MD5_EXPORT
RC2_MD5_EXPORT
```

For more information about setting the ciphers to be used, see `ldap_get/set_option()`.

To specify the number of seconds for the SSL session-level timer, use:

```
ldap_set_option(ld,LDAP_OPT_SSL_TIMEOUT, &timeout)
```

where timeout specifies timeout in seconds. When timeout occurs, SSL again establishes the session keys for the session, for increased security. To specify a specific cipher, or set of ciphers, to be used when you negotiate with the server, use `ldap_set_option()` to define a sequence of ciphers. For example, the following setting defines a sequence of three ciphers to be used when you negotiate with the server. The first cipher that is found to be in common with the server list of ciphers is used.

`ldap_set_cipher` is the same as calling `ldap_set_option` (`ld`, `LDAP_OPT_SSL_CIPHER`, option). Either function checks the validity of the input string. The cipher is used when the SSL connection is established by `ldap_ssl_init()`. For more information about `ldap_set_option`, see "LDAP_INIT" on page 63.

`ldap_ssl_set_fips_mode_np()` can be called before you call `ldap_ssl_environment_init()` or `ldap_ssl_client_int()` to set FIPS processing mode. If FIPS processing mode is supposed to be on, SSL uses the FIPS certified encryption libraries for encryption and sets the processing mode to `on`. FIPS processing mode does not change any existing SSL environments.

Options

Options are supported for controlling the nature of the secure connection. These options are set by using the `ldap_set_option()` API.

```
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER,
(void *) LDAP_SSL_3DES_SHA_US
LDAP_SSL_RC4_MD5_US);
```

The following ciphers are defined in `ldap.h`:

```
#define LDAP_SSL_RC4_SHA_US "05"
#define LDAP_SSL_RC4_MD5_US "04"
#define LDAP_SSL_DES_SHA_US "09"
#define LDAP_SSL_3DES_SHA_US "0A"
#define LDAP_SSL_RC4_MD5_EX "03"
#define LDAP_SSL_RC2_MD5_EX "06"
```

For more information about `ldap_set_option`, see "LDAP_INIT" on page 63.

Notes

`ldapssl.h` contains return codes that are specific for `ldap_ssl_client_init()`, `ldap_ssl_init()`, and `ldap_ssl_start()`.

The SSL versions of these utilities include RSA Security Inc. software.

The `ldap_ssl_client_init()`, `ldap_ssl_init()`, and `ldap_ssl_start()` APIs are only supported for the versions of the LDAP library that include the SSL component.

`ldap_ssl_set_fips_mode_np()` returns `LDAP_SUCCESS` if the client library supports SSL, otherwise it returns `LDAP_SSL_NOT_AVAILABLE`.

See also

`ldap_init`, `ldap_ssl_environment_init`, and `ldap_ssl_client_init`

LDAP_SSL_PKCS11

Use the `LDAP_SSL_PKCS11` API or LDAP routine for setting up the SSL and `PKCS#11` environment for `GSKit`.

- `ldap_ssl_pkcs11_client_init`
- `ldap_ssl_pkcs11_environment_init`

Synopsis

```
#include ldap.h
#include ldapssl.h


typedef  struct {
         char       *Libpath,
         char       *TokenLabel,
         char       *TokenPw,
         int        Keystorage,
         int        Accelerator} PKCS11arg;

int ldap_ssl_pkcs11_client_init(
         char       *keydatabase,
         char       *keydatabase_pwd,
         int        ssl_timeout,
         int        *pSSLReasonCode,
         PKCS11arg  *pkcs11arg);

ldap_ssl_pkcs11_environment_init(
         char       *keydatabase,
         char       *keydatabase_pwd,
         int        ssl_timeout,
         int        *pSSLReasonCode,
         PKCS11arg  *pkcs11arg);
```

Input parameters

keydatabase

Specifies the name of the key database file with `kdb` extension. The key database file typically contains one or more certificates from the certificate authorities (CAs) that are trusted by the clients. If the LDAP server is configured to provide only server authentication, then a private key and client certificate are not required. If the user wants to use the Crypto device under key storage mode only, then the **`keydatabase`** parameter can be NULL. If the client needs the Crypto device to work only in accelerator mode, then the `kdb` file must be specified. If the key database file and password are NULL, then the default `ldapkey.kdb` file is used as the key database and the password is used from default `ldapkey.sth` file.

User is given a provision to store some keys on device. This provision can be Personal Certificates with private key, and some in the key database file, which can be Signer Certificates with public keys. Therefore, a specific certificate is selected either from the local `kdb` file or from Crypto device that is based on the certificate label used.

keydatabase_pwd
Specifies the password that is used to protect the contents of the key database file. This password is important, particularly when it protects one or more private keys that are stored in the key database file. If NULL is passed to this parameter and the key database file is NULL, then password for the default `ldapkey.kdb` file is taken from `ldapkey.sth` file.

ssl_timeout
Specifies the SSL timeout value in seconds. The timeout value controls the frequency with which the SSL protocol stack regenerates session keys. If **`ssl_timeout`** is set to 0, then the default value 43,200 is used. Otherwise, the value specified in the parameter is used, this value should be less than or equal to 86,400 (number of seconds in a day). If **`ssl_timeout`** is greater than 86,400, then `LDAP_PARAM_ERROR` is returned.

pSSLReasonCode
Specifies a pointer to the SSL reason code that contains more information in event of an error occurs during the initialization of the SSL stack. See `ldapssl.h` for reason codes that can be returned.

pkcs11arg
Specifies a struct data type that contains information about the different Crypto device settings to enable key storage and accelerator mode.

An instance of a structure contains following fields:

Libpath
Specifies a null terminated string that defines the driver path of the file that is to be used to access `PKCS11` device.

Token_label
Specifies a null terminated string that defines the label that is assigned to the `PKCS11` device for access.

Token_pwd
Specifies a null terminated string that defines the password phrase to access the `PKCS11` device.

Keystorage
The value of this parameter can be 0 or 1. If set to 1, it indicates that the Crypto device is to be used as key storage. If set to 0, it indicates that the Crypto device cannot function as key storage.

Accelerator
Specifies an integer value that determines the type of accelerated operation a client needs from the `PKCS11` device.

Under acceleration mode, the `PKCS11` device can be configured to do three different operations: Symmetric operation, Digest operation, and Random Data Generation operation.

The accelerator value must be one of the options that are listed as follows:

```
#define LDAP_SSL_ACCELERATION_MODE_NONE         0
#define LDAP_SSL_ACCELERATION_MODE_SYM          1
#define LDAP_SSL_ACCELERATION_MODE_DIG          2
```

```
#define LDAP_SSL_ACCELERATION_MODE_SYM_DIG       3
#define LDAP_SSL_ACCELERATION_MODE_RND           4
#define LDAP_SSL_ACCELERATION_MODE_RND_SYM       5
#define LDAP_SSL_ACCELERATION_MODE_RND_DIG       6
#define LDAP_SSL_ACCELERATION_MODE_SYM_DIG_RND   7
```

Usage

The `ldap_ssl_pkcs11_client_init()` routine is used to initialize the SSL and PKCS#11 environment for an application process. For every application process, the `ldap_ssl_pkcs11_client_init()` API must be called before its makes any other SSL-related LDAP calls, such as `ldap_ssl_init()`. When `ldap_ssl_pkcs11_client_init()` is successfully called, any subsequent invocations return a return code of `LDAP_SSL_ALREADY_INITIALIZED`. This call indicates that a particular key database file and PKCS#11 settings are effectively bound to an application process. To change the SSL configuration in use, the application or one of its processes must be restarted.

The `ldap_ssl_pkcs11_environment_init()` routine can be used instead of `ldap_ssl_pkcs11_client_init()`. This API can be called more than one time in the same process, where each call creates an SSL environment that is used for subsequent SSL sessions that are initiated by calling `ldap_ssl_init()`. In these cases, the SSL environments persist until the LDAP sessions that were created by using them persist.

See also

`ldap_ssl_environment_init`, `ldap_ssl_client_init`, `ldap_ssl_init`

LDAP_START_TRANSACTION

Use the `LDAP_START_TRANSACTION` API or LDAP routine to call a start transaction request.

- `ldap_start_transaction`
- `ldap_start_transaction_s`

Synopsis

```
#include ldap.h


int ldap_start_transaction(
       LDAP           *ld,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

int ldap_start_transaction_s(
       LDAP           *ld,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       struct berval  **retdatap)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

serverctrls
Specifies a list of LDAP server controls.

clientctrls
Specifies a list of LDAP client controls.

Output parameters

msgidp
This parameter contains the message ID of the request.

retdatap
This parameter specifies the result of the start transaction operation. This result contains the transaction ID of the transaction.

Usage

This API routine is used to initiate a start transaction request against the server.

Errors

This API routine returns an LDAP error code if the operation is unsuccessful.

See also

ldap_start_transaction, ldap_start_transaction_s, ldap_prepare_transaction, ldap_prepare_transaction_s, ldap_end_transaction, ldap_end_transaction_s, ldap_get_tran_id, ldap_create_transaction_control

LDAP_START_TLS

Use the `LDAP_START_TLS` API or LDAP routine to start a `TLS` session.

```
ldap_start_tls_s_np
```

Synopsis

```
#include ldap.h

int ldap_start_tls_s_np (
 LDAP  *ld,
 const char *certificateName)
```

Input parameters

ld Specifies the LDAP pointer that is used in the `ldap_start_tls_s_np()` call.

certificateName
Specifies the name of the certificate to use. It is the same as the parameter used in the `ldap_ssl_environment_init()` API and might be NULL.

Usage

The `ldap_start_tls_s_np()` API is used to secure a previously unsecured connection. It takes a handle from an existing LDAP connection and the name of the certificate to use. If the command is successful, then communication on the connection is secure until either the connection is closed or an `ldap_stop_tls_s_np()` call is made.

The secure environment must be initialized, either by calling `ldap_ssl_environment_init` or `ldap_ssl_client_init`, before `ldap_start_tls_s_np()` is called.

Errors

ldap_start_tls_s_np() returns LDAP_SUCCESS if the call was successful, or an LDAP error if the call was unsuccessful.

If the connection is already secure, either by going against the SSL port or by already establishing a TLS session, then LDAP_OPERATIONS_ERROR is returned.

If the secure environment is not initialized through a call to ldap_ssl_client_init or ldap_ssl_environment_init, then LDAP_TLS_CLIENT_INIT_NOT_CALLED is returned.

If the TLS handshake with the server fails, LDAP_TLS_HANDSHAKE_FAILED is returned.

If the server is not configured to allow TLS, then LDAP_PROTOCOL_ERROR is returned.

If the GSKit environment was not previously initialized, then LDAP_SSL_CLIENT_INIT_NOT_CALLED is returned.

If the server does not support TLS, then LDAP_REFERRAL is returned. The referred to server might support TLS.

ldap_stop_tls_s_np, ldap_ssl_environment_init, ldap_ssl_client_init

See also

If the server is configured to do TLS, but is unable to establish TLS connections, then LDAP_UNAVAILABLE is returned.

LDAP_STOP_TLS

Use the LDAP_STOP_TLS API or LDAP routine to abandon an open LDAP connection over TLS.

```
ldap_stop_tls_s_np
```

Synopsis

```
#include ldap.h

int ldap_stop_tls_s_np(
 LDAP  *ld)
```

Input parameters

ld Specifies the LDAP pointer that is used in the ldap_start_tls_s_np() call.

Usage

The ldap_stop_tls_s_np() API is used to end the TLS session on a connection.

This call closes the connection to the server.

Errors

ldap_stop_tls_s_np() returns LDAP_SUCCESS if the call was successful, an LDAP error if the call was unsuccessful.

See also

ldap_start_tls_s_np, ldap_ssl_environment_init, ldap_ssl_client_init

LDAP_URL

Use the `LDAP_URL` API for Uniform Resource Locator routines.

```
ldap_is_ldap_url
ldap_url_parse
ldap_free_urldesc
ldap_url_search
ldap_url_search_s
ldap_url_search_st
```

Synopsis

```
#include sys/time.h /* for struct timeval definition */

#include ldap.h


int ldap_is_ldap_url(
       char          *url)

int ldap_url_parse(
       char          *url,
       LDAPURLDesc    **ludpp)

typedef struct ldap_url_desc {
    char    *lud_host;      /* LDAP host to contact */
    int      lud_port;      /* port on host */
    char    *lud_dn;        /* base for search */
    char   **lud_attrs;     /* NULL-terminate list of attributes */
    int      lud_scope;     /* a valid LDAP_SCOPE_... value */
    char    *lud_filter;    /* LDAP search filter */
    char    *lud_string;    /* for internal use only */
} LDAPURLDesc;

ldap_free_urldesc(
       LDAPURLDesc    *ludp)

int ldap_url_search(
       LDAP          *ld,
       char          *url,
       int           attrsonly)

int ldap_url_search_s(
       LDAP          *ld,
       char          *url,
       int           attrsonly,
       LDAPMessage    **res)

int ldap_url_search_st(
       LDAP          *ld,
       char          *url,
       int           attrsonly,
       struct timeval *timeout,
       LDAPMessage    **res)
```

Input parameters

ld Specifies the LDAP pointer that is returned by a previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`.

url Specifies a pointer to the URL string.

attrsonly
Specifies attribute information. Set to 1 to request attribute types only. Set to 0 to request both attribute types and attribute values.

timeout
Specifies a timeout value for a synchronous search that is issued by the `ldap_url_search_st()` routine.

ludp Points to the LDAP URL description, as returned by `ldap_url_parse()`.

Output parameters

ludpp Points to the LDAP URL description, as returned by `ldap_url_parse()`.

res Contains the result of the asynchronous operation that is identified by `msgid`, as returned from `ldap_url_search_s()` or `ldap_url_search_st()`. This result must be passed to the LDAP parsing routines.

Usage

These routines support the use of LDAP URLs. LDAP URLs look like the following format:

`ldap://[`*`hostname`*`]/`*`dn`*`[?`*`attributes`*`[?`*`scope`*`[?`*`filter`*`]]]`

where:
- *hostname* is a host name with an optional: *portnumber*.
- *dn* is the base DN to be used for an LDAP search operation.
- *attributes* is a comma-separated list of attributes to be retrieved.
- *scope* is one of the following three strings: base, one, or sub. The default is base.
- *filter* is the LDAP search filter as used in a call to `ldap_search`.

For example:

```
ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
```

URLs that are wrapped in angle-brackets or preceded by `URL:` or both are also tolerated, including the following forms:
- `URL:ldapurl`

  For example:

  ```
  URL:ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
  ```
- `URL:ldapurl`

  For example:

  ```
  URL:ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
  ```

`ldap_is_ldap_url()` returns a nonzero value if URL begins with `ldap://`. It can be used as a quick check for an LDAP URL; the `ldap_url_parse()` routine is used to extract the various components of the URL.

`ldap_url_parse()` breaks down an LDAP URL passed in URL into its component pieces. If successful, zero is returned, an LDAP URL description is allocated and provided, and **`ludpp`** is set to point to it. If an error occurs, one of these values is returned:

```
LDAP_URL_ERR_NOTLDAP    - URL doesn't begin with "ldap://"
LDAP_URL_ERR_NODN   - URL has no DN (required)
LDAP_URL_ERR_BADSCOPE   - URL scope string is invalid
LDAP_URL_ERR_MEM    - can't allocate memory space
```

ldap_free_urldesc() is called to free an LDAP URL description that was obtained from a call to ldap_url_parse().

ldap_url_search() initiates an asynchronous LDAP search that is based on the contents of the URL string. This routine acts just like ldap_search except that the search parameters are pulled out of the URL.

ldap_url_search_s() does a synchronous LDAP search that is based on the contents of the URL string. This routine acts just like ldap_search_s() except that the search parameters are pulled out of the URL.

ldap_url_search_st() does a synchronous LDAP URL search with a specified timeout. This routine acts just like ldap_search_st() except that the search parameters are pulled out of the URL.

Notes

For search operations, if **hostport** is omitted, host and port for the current connection are used. If **hostport** is specified, and is different from the host and port combination that is used for the current connection, the search is directed to **hostport**, instead of using the current connection. In this case, the underlying referral mechanism is used to bind to **hostport**.

If the LDAP URL does not contain a search filter, the filter defaults to objectClass=*.

See also

ldap_search

LDAP_SSL_ENVIRONMENT_INIT

Use the LDAP_SSL_ENVIRONMENT_INIT API to initialize SSL for a secure connection between a client and server.

The ldap_ssl_environment_init() routine has the same parameters as ldap_ssl_client_int() but can be called more than one time. It returns LDAP_SUCCESS or the appropriate LDAP error code. It does not return LDAP_SSL_ALREADY_INITIALIZED. An application that requires SSL connections to different servers can initialize environments in separate calls to this function, with different key database files. The environment that is created is used by all SSL connections that are established by calling ldap_ssl_init() until the next call is made to ldap_ssl_environment_init(). Subsequent calls to ldap_ssl_environment_init() do not affect existing SSL connections.

Note: This routine is deprecated but is still supported.

Synopsis

```
#include ldap.h
#include ldapssl.h
int ldap_ssl_environment_init(
 const char       *keydatabase,
 const char       *keydatabase_pw,
 int              ssl_timeout,
 int              *pSSLReasonCode);
```

Input parameters

keydatabase
: Specifies the name of a key database file with `.kdb` extension. The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. A key database can be used to store the private keys of the client and associated client certificates. A private key and associated client certificate are required if the LDAP server is configured to require client and server authentication only. If the LDAP server is configured to provide only server authentication, a private key and client certificate are not required.

keydatabase_pw
: Specifies the password that is used to protect the contents of the key database. This password is important, particularly when it protects one or more private keys that are stored in the key database. The password is specified when the key database is initially created, and can be changed by using the **`iKeyman`** utility. Instead of specifying the password each time the application opens the key database. The password can be obtained from a password stash file that contains an encrypted version of the password. The password stash file can be created by using the **`iKeyman`** utility. To obtain the password from the password stash file, specify a NULL pointer for `keydatabase_pw`. It is assumed that the password stash file has the same name as the key database file, but with a `.sth` extension instead of `.kdb`. It is assumed that the password stash file is in the same directory as the key database file.

  Note: The default key database file, `ldapkey.kdb`, is initially configured to have `ssl_password` as its password. This password is also initially configured in the default password stash file (`ldapkey.sth`).

ssl_timeout
: Specifies the SSL timeout value in seconds. The timeout value controls the frequency with which the SSL protocol stack regenerates session keys. If **`ssl_timeout`** is set to 0, a default value is used. Otherwise, the value that is supplied is used, provided it is less than or equal to 86,400, the number of seconds in a day. If **`ssl_timeout`** is greater than 86,400, `LDAP_PARAM_ERROR` is returned.

pSSLReasonCode
: Specifies a pointer to the SSL Reason Code, which provides more information when an error occurs during initialization of the SSL stack, when `ldap_ssl_environment_init()` is called. See `ldapssl.h` for reason codes that can be returned.

See also

ldap_ssl_pkcs11_client_init, ldap_ssl_pkcs11_environment_init

LDAP_SORT

Use the `LDAP_SORT` API or LDAP routine to request the entries that are sort and returned by the servers that match the filter that is specified on a search operation.

```
ldap_create_sort_keylist
ldap_free_sort_keylist
ldap_create_sort_control
```

```
ldap_parse_sort_control
```

Synopsis

```
#include ldap.h

int ldap_create_sort_keylist(
   LDAPsortkey      ***sortKeyList,
   const char       *sortString);

int ldap_create_sort_control(
   LDAP             *ld,
   LDAPsortkey      **sortKeyList,
   const char       isCritical,
   LDAPControl      **control)

void ldap_free_sort_keylist(
   LDAPsortkey      **sortKeyList)

int ldap_parse_sort_control(
   LDAP             *ld,
   LDAPControl      **serverControls,
   unsigned long    *sortRC,
   char             **attribute)
```

Input parameters

ld
Specifies the LDAP pointer that is returned by previous call to `ldap_init()`, `ldap_ssl_init()`, or `ldap_open()`. Must not be NULL.

sortString
String with one or more attributes to be used to sort entries that are returned by the server.

sortKeyList
Pointer to an array of `LDAPsortkey` structures, which represent attributes that the server uses to sort returned entries. Input when used for `ldap_create_sort_control()` and `ldap_free_sort_keylist()`.

isCritical
Specifies the criticality of sort on the search. If the criticality of sort is `FALSE`, and the server finds a problem with the sort criteria, the search continues. However, the entries that are returned are not sorted. If the criticality of sort is TRUE, and the server finds a problem with the sort criteria, the search does not continue. No sorting is done, and no entries are returned. If the server does not find any problem with the sort criteria, the search and sort continues and entries are returned sorted.

serverControls
A list of LDAP server controls. For more information about server controls, see “LDAP controls” on page 27. These controls are returned to the client when you call the `ldap_parse_result()` function on the set of results that are returned by the server.

Output parameters

sortKeyList
Pointer to an array of `LDAPsortkey` structures, which represent attributes the server uses to sort returned entries. Output when used for `ldap_create_sort_keylist()`.

control
A result parameter that is provided with an allocated array of one control for the sort function. The control must be freed by calling `ldap_control_free()`.

sortRC
LDAP return code that is retrieved from the sort results control returned by the server.

attribute
Returned by the server, it is the name of the attribute in error.

Usage

These routines are used to sort the entries that are returned from the server after an LDAP search operation.

The `ldap_create_sort_keylist()` function builds a list of `LDAPsortkey` structures that are based on the list of attributes that are included in the incoming string. A sort key is made up of three possible values:

- Name of attribute that is used to sort entries that are returned by the server
- OID of a matching rule for that attribute
- Whether the sort must be done in reverse order

The syntax of the attributes in the `sortString`, `[-]`*`attribute name`*`[:<matching rule OID>]`, specifies whether there is a matching rule OID that must be used for the attribute, and whether the attribute must be sorted in reverse order. In the following `sortString` example, the search results are sorted first by surname and then by given name, with the given name that is sorted in reverse (descending order) as specified by the prefixed minus sign `( - )`:

```
sn -givenname
```

Thus, the syntax of the sort parameter is as follows:

```
[-]attribute name[:matching rule OID]
```

where:

- *attribute name* is the name of the attribute you want to sort by.
- *matching rule OID* is the optional OID of a matching rule that you want to use for sorting.
- the minus sign `( - )` indicates that the results must be sorted in reverse order.

The `sortKeyList`, output from the `ldap_create_sort_keylist()` function, can be used as input into the `ldap_create_sort_control()` function. The `sortKeyList` is an ordered array of `LDAPsortkey` structures such that the key with the highest precedence is at the front of the array. The control output form `ldap_create_sort_control()` function includes the criticality set based on the value of the `isCritical` flag. This control is added to the list of client controls that are sent to the server on the LDAP search request.

The `ldap_free_sort_keylist()` function cleans up all the memory that is used by the sort key list. This function must be called after the `ldap_create_sort_control()` function is completed.

When a sort results control is returned by the server, the `ldap_parse_sort_control()` function can be used to retrieve the values from the control. The function takes as input the server controls returned by the server, and returns the value of the sort control return code and possibly an attribute name if the return code is not LDAP_SUCCESS. No sort control is returned to the client if:

- There was an error that parses the sort criteria for the search.
- There were no entries that are returned for the search.

Server-side sorting of search results

Sorted Search Results provides sort capabilities for LDAP clients with limited or no sort functions. Sorted Search Results enables an LDAP client to receive sorted search results that are based on a list of criteria. Each criteria represents a sort key. The sort criteria includes attribute types, matching rules, or descending order. The server must use this criteria to sort search results before you return them. This criteria moves the responsibility of sorting from the client application to the server, where it might be done much more efficiently. For example, a client application might want to sort the list of employees at their Grand Cayman site by surname, common name, and telephone number. Instead of building the search list twice for sorting, the search list is built one time. It is then sorted, before it returns the results to the client application. The sort is done one time at the server and then again at the client when all the results are returned.

In the following sortString example, the search results are sorted first by surname (sn), then by given name (givenname), with the given name that is being sorted in reverse (descending) order as specified by the prefixed minus sign (-).

```
sn -givenname
```

The sortKeyList output from ldap_create_sort_keylist() can be used as input to ldap_create_sort_control(). The sortKeyList is an ordered array of LDAPsortkey structures such that the key with the highest precedence is at the front of the array. ldap_create_sort_control() outputs a LDAPControl structure which can be added to the list of client controls that are sent to the server on the LDAP search request. The LDAPControl structure that is returned by the ldap_create_sort_control() API can be used as input to ldap_search_ext() or ldap_search_ext_s(), which are used to make the actual search request.

Note: Server-side sorting is an optional extension of the LDAP v3 protocol, so the server you are bound to before the ldap_search_ext() or ldap_search_ext_s() call might not support this function.

Now that you created the server-side control, you can free the sortKeyList output from ldap_create_sort_keylist() by using ldap_free_sort_keylist().

Upon completion of the search request that you submitted by using ldap_search_ext() or ldap_search_ext_s(), the server returns an LDAP result message that includes a sort results control. The client application can parse this control by using ldap_parse_sort_control() which takes the returned server response controls (a null terminated array of pointers to LDAPControl structures) as input. ldap_parse_sort_control() outputs a return code that indicates whether the sort request was successful. If the sort was not successful, the name of the attribute in error might be output from ldap_parse_sort_control(). Use ldap_controls_free() to free the memory that is used by the client application to hold the server controls when you are done processing all controls that are returned by the server for this search request.

The server returns a successful return code of LDAP_SUCCESS in the sort response control (sortKeyResponseControl) in the search result (searchResultDone) message if the server supports sorting and can sort the search results by using the specified keys. If the search fails for any reason or there are no search results, then the server omits the sortKeyResponseControl from the searchResultsDone message.

If the server does not support sorting and the criticality that is specified on the sort control for the search request is TRUE, the server does not return any search results, and the sort response control return code is set to LDAP_UNAVAILABLE_CRITICAL_EXTENSION. If the server does not support the sorting and the criticality that is specified on the sort control for the search request is FALSE, the server returns all search results. The sort control is ignored.

If the server does support the sorting and the criticality that is specified on the sort control for the search request is TRUE, but for some reason the server cannot sort the search results, then the sort response control return code is set to LDAP_UNAVAILABLE_CRITICAL_EXTENSION and no search results are returned. If the server does support the sorting and the criticality that is specified on the sort control for the search request is FALSE, and for some reason the server cannot sort the search results, then the sort response control return code is set to the appropriate return code and all search results are returned unsorted.

The following return codes might be returned by the server in the sortKeyResponseControl of the searchResultDone message:

- LDAP_SUCCESS - the results are sorted
- LDAP_OPERATIONS_ERROR - server internal failure
- LDAP_TIMELIMIT_EXCEEDED - time limit that is reached before the sorting was completed
- LDAP_STRONG_AUTH_REQUIRED - refused to return sorted results by using insecure protocol
- LDAP_ADMIN_LIMIT_EXCEEDED - too many matching entries for the server to sort
- LDAP_NO_SUCH_ATTRIBUTE - unrecognized attribute type in sort key
- LDAP_INAPPROPRIATE_MATCHING - unrecognized or inappropriate matching rule in sort key
- LDAP_INSUFFICIENT_ACCESS - refused to return sorted results to this client
- LDAP_BUSY - too busy to process
- LDAP_UNWILLING_TO_PERFORM - unable to sort
- LDAP_OTHER - unable to sort due to reasons other than the reasons specified earlier

There are other rules that must be considered when you request sort from the server. These rules are as follows:

- The matching rule must be one that is valid for the sort attribute it applies to. The server returns LDAP_INAPPROPRIATE_MATCHING if it is not.
- If the matching rule is omitted from a sort key, the ordering matching rule that is defined for use with this sort attribute must be used.
- A server can restrict the number of keys that are supported for a sort control, such as supporting only one key. (A sort key list of at least one key must be supported).

- A search result meets the search criteria but is missing a value for the sort key (sort attribute value is NULL). Then, this search result is considered a larger value than any other valid values for that key.

When sorted search is requested along with simple paged results, the `sortKeyResponseControl` is returned on every `searchResultsDone` message, not just the last one of the paged results requests. The `sortKeyResponseControl` might not be returned if there is an error that processes the paged results request or there are no search results to return. When sorted search is requested along with simple paged results, the server sends the search results sorted based on the entire search result set. It does not sort each page. For more information, see the section *Simple paged results of search results* in "LDAP_PAGED_RESULTS" on page 85.

When it chases referrals, the client application must send in a sorted search request to each of the referral servers. It is up to the application that uses the client services to decide whether to set the criticality as to the support of sorted search results. It is also to handle a lack of support of this control on referral servers as appropriate based on the application. Additionally, the LDAP server does not ensure that the referral server supports the sorted search control. Multiple lists might be returned to the client application, some of which are not sorted. It is the decision of the client application as to how best to present this information to the user. Possible solutions include:

- Combine all referral results before it presents to the user
- Show multiple lists and the corresponding referral server host name
- Take no extra steps and show all results to the user as they are returned from the server

The client application must turn off referrals to get one truly sorted list. Otherwise, when it chases referrals with the sorted search control specified, unpredictable results can occur.

More information about the server side sorted search control, with control OID of 1.2.840.113556.1.4.473, can be found in *RFC 2891 - LDAP Control Extension for Server Side Sorting of Search Results*.

Errors

The sort routines return an LDAP error code if they encounter an error that parses the result. See "LDAP_ERROR" on page 45 for a list of the LDAP error codes.

Notes

`SortString`, `sortKeyList`, controls, `serverControls`, and attribute must be freed by the caller.

See also

ldap_search, ldap_parse_result

LDAP_SSL_SET_EXTN_SIGALG

Use the `ldap_ssl_set_extn_sigalg` API in an LDAP application to set `TLS 1.2` signature and hash algorithms.

Purpose

You can use the `ldap_ssl_set_extn_sigalg()` API to set `TLS 1.2` signature and hash algorithm restriction in an LDAP client environment. You must call this API before you issue a bind or any other operations that connects to the server.

Synopsis

```
#include <ldap.h>

int ldap_ssl_set_extn_sigalg(
                char *sigalg)
```

Input parameters

`sigalg`
Specifies the address of the values to set with `ldap_ssl_set_extn_sigalg()`.

Session settings

In IBM Security Directory Server, version 6.3, Fix Pack 17 or later, you can restrict a client utility to communicate with the `TLS 1.2` protocol and a set of `TLS 1.2` signature and hash algorithm. Set the required `TLS 1.2` signature and hash algorithm values by calling the `ldap_ssl_set_extn_sigalg()` API before the gsk_envrionment_init call initializes the GSKit environment.

You must pass the **`sigalg`** parameter to `ldap_ssl_set_extn_sigalg()` and set the required `TLS 1.2` signature and hash algorithm. The `ldap_ssl_set_extn_sigalg()` API sets the `TLS 1.2` signature and hash algorithm restriction for the LDAP connection that is prepared after this call. The following values are supported by the **`sigalg`** parameter:

```
GSK_TLS_SIGALG_RSA_WITH_SHA224
GSK_TLS_SIGALG_RSA_WITH_SHA256
GSK_TLS_SIGALG_RSA_WITH_SHA384
GSK_TLS_SIGALG_RSA_WITH_SHA512
GSK_TLS_SIGALG_ECDSA_WITH_SHA224
GSK_TLS_SIGALG_ECDSA_WITH_SHA256
GSK_TLS_SIGALG_ECDSA_WITH_SHA384
GSK_TLS_SIGALG_ECDSA_WITH_SHA512
```

Errors

If an error occurs, the `TLS 1.2` signature and hash algorithm restriction is not set. To obtain a detailed error report, you must run the application in debug mode and check the debug traces.

Examples

To configure `TLS 1.2` signature and hash algorithm restriction for an LDAP client, you can use the following example.

```
char *sigalg=NULL;

if (ssl == 1){
      if (sigalg != NULL){
         ldap_ssl_set_extn_sigalg(sigalg);
      }
   }
```

LDAP_SSL_SET_SUITEB_MODE

Use the `ldap_ssl_set_suiteb_mode` API in an LDAP application to set Suite B mode.

Purpose

You can use the `ldap_ssl_set_suiteb_mode()` API in LDAP client utilities to set a Suite B mode value in a client environment. When you set Suite B mode in a client utility, the API sets Suite B mode internally in the GSKit environment. You must call this API before you issue a bind or any other operations that connects to the server.

Synopsis

```
#include <ldap.h>

int ldap_ssl_set_suiteb_mode(
             char *suitebEnv)
```

Input parameters

`suitebEnv`
Specifies the address of the value to set with `ldap_ssl_set_suiteb_mode()`.

Session settings

In IBM Security Directory Server, version 6.3, Fix Pack 17 or later, you can set Suite B mode in an LDAP client with the `ldap_ssl_set_suiteb_mode()` API. You must call the `ldap_ssl_set_suiteb_mode()` API with an appropriate value to set Suite B mode before you initialize GSKit environment with the gsk_envrionment_init call.

You can set Suite B mode after the ICC version of the libraries is set in the GSKit environment. You must also set the FIPS-140 certified cryptographic modules in the GSkit environment. The `ldap_ssl_set_suiteb_mode()` API internally configures the cryptographic modules. The `ldap_ssl_set_suiteb_mode()` API might fail if the settings cannot be configured in the GSKit environment.

You must pass the **`LDAP_OPT_SUITEB_MODE`** parameter to the `ldap_ssl_set_suiteb_mode()` API to set Suite B mode. The following values are valid for the **`LDAP_OPT_SUITEB_MODE`** parameter:

- `128`
- `192`

Before you prepare an LDAP connection, you must call the `ldap_ssl_set_suiteb_mode()` API and set the Suite B environment.

After you set Suite B mode in an LDAP client environment, you must connect to the secure port of a directory server with an LDAP client.

Note: An LDAP client uses the `TLS 1.2` protocol to secure communication with a directory server. All other protocols, such as `SSLv3`, `TLS 1.0`, and `TLS 1.1` are not supported.

Errors

If an error occurs, Suite B mode is not set. To obtain a detailed error report, you must run the application in debug mode and check the debug traces.

Examples

To configure Suite B mode for an LDAP client, use the following example:

```
char *suitebEnv=NULL;
if (ssl == 1){
      suitebEnv = getenv("LDAP_OPT_SUITEB_MODE");
      if (suitebEnv != NULL){
         ldap_ssl_set_suiteb_mode(suitebEnv);
      }
   }
```

Chapter 3. IBM Security Directory Server Java Naming and Directory Interface (JNDI) Toolkit

IBM Security Directory Server provides Java Naming and Directory Interface (JNDI) Toolkit that contains Java classes for most of the extended operations and controls in IBM Security Directory Server.

Implementing extended operations by using IBM Security Directory Server JNDI Toolkit

An extended operation is a mechanism that allows more operations that are not defined in the LDAP protocol to be supported for services. These services are provided by LDAP V3 servers.

All additional operations that a server supports are to be sent by the client as an extended operation. An extended operation is called when a client sends an extended request and receives an extended response in response from the server. This communication between the client and server is broadly sequences as:

1. A client sends an extended request to the server.
2. If the server recognizes the request, it runs the operation.
3. An extended response is sent back with the result, if any, for the operation.

When an extended operation is implemented by using JNDI, each extended operation has a request class and a response class. An extended request is made of two parts:

requestName
: The **requestName** field contains a unique dotted decimal representation of the OID (Object Identifier) that identifies the request. The OID namespace is hierarchically divided, every authority that can define an OID is assigned a prefix that it uses to identify its OID.

requestValue
: The **requestValue** field contains data that is needed to run the request. The format of the data is predefined for every extended operation. Some extended operations do not require any data to be associated with a request.

The extended request is encoded before it is sent to the server. IBM Security Directory Server requires the extended request to be ASN.1 BER encoded.

The **javax.naming.ldap.ExtendedRequest** interface describes an extended operation request. This interface contains methods `getID()` and `getEncodedValue()` that retrieve the two properties of an extended operation request, `requestName` and `requestValue`. A request class implements the **javax.naming.ldap.ExtendedRequest** interface and overrides the following methods of this interface `public String getID()` and `public byte[] getEncodedValue()`. This method retrieves ASN.1 BER encoded value from the LDAP extended operation request and constructs request sequence for the extended operation by using the `com.ibm.asn1.BEREncoder` class. This method then converts the encoded request sequence to byte array to be returned by the method.

The `createExtendedResponse` method creates the response object corresponding to a request. When a caller sends the extended operation request to the LDAP server, a response from the server is sent back. If the operation fails, the caller throws the `NamingException` exception. If the operation succeeds, the caller calls this method by using the data that it received in the response. The purpose of this method is to return an object of class that implements the **ExtendedResponse** interface that is appropriate for the extended operation request.

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

The parameters that are passed to this method:

id An object identifier of the response control.

berValue
An ASN.1 BER encoded value of the response control. This value is the raw BER bytes including the tag and length of the response value. It does not include the response OID.

offset The starting position in the `berValue` of the bytes to use.

length The number of bytes to use from `berValue`.

The structure of the extended response is similar to extended request. It can contain the OID and value both of which are optional, and a field that describes the result code of the operation. An extended response is made of three parts:

resultcode
The **resultcode** field contains result code of the operation. The result code can contain one of the defined LDAP error codes. For example, `LDAP_SUCCESS` and `LDAP_OPERATIONS_ERROR`.

responseName
The **responseName** field contains OID of the response. It might or might not be same as the request OID.

responseValue
The **responseValue** field contains the result of the operation, if any.

The response from the server is encoded in ASN.1 BER code and must be decoded by the client when received.

The **javax.naming.ldap.ExtendedResponse** interface describes an extended operation response. A response class implements the **javax.naming.ldap.ExtendedResponse** interface. The methods in this class can be used by the application to get low-level information about the extended operation response. This class parses the extended response and provides methods specific to that extended operations to return response values. It uses **`com.ibm.asn1.BERDecoder`** class to parse the response from the LDAP server. The **`BEREncoder`** and **`BERDecoder`** classes are part of current `IBMLDAPJavaBer.jar` shipped with IBM Security Directory Server.

The Java classes for extended operations that are provided in IBM Security Directory Server JNDI Toolkit are listed.

Table 5. Java classes for extended operations provided in IBM Security Directory Server JNDI Toolkit

Extended operations	Java classes	Request OID or Response OID
Account status	AccountStatusRequest AccountStatusResponse	1.3.18.0.2.12.58/ 1.3.18.0.2.12.59
Attribute type	GetAttributesRequest GetAttributesResponse	1.3.18.0.2.12.46 / 1.3.18.0.2.12.47
Begin transaction	TransactionStartRequest TransactionStartResponse	1.3.18.0.2.12.5
Cascading replication operation	CascadingReplicationRequest CascadingReplicationResponse	1.3.18.0.2.12.15
Clear log	ClearLogRequest ClearLogResponse	1.3.18.0.2.12.20 / 1.3.18.0.2.12.21
Control replication	ControlReplicationRequest ControlReplicationResponse	1.3.18.0.2.12.16
Control queue	ControlQueueRequest ControlQueueResponse	1.3.18.0.2.12.17
DN normalization	NormalizeDNRequest NormalizeDNResponse	1.3.18.0.2.12.30
Dynamic server trace	ControlTracingRequest ControlTracingResponse	1.3.18.0.2.12.40
Dynamic update requests	ReadConfigurationRequest ReadConfigurationResponse	1.3.18.0.2.12.28 / 1.3.18.0.2.12.29
End transaction	TransactionEndRequest TransactionEndResponse	1.3.18.0.2.12.6
Effective password policy	EffectivePwdPolicyRequest EffectivePwdPolicyResponse	1.3.18.0.2.12.75/ 1.3.18.0.2.12.77
Event notification register request	RegisterEventRequest RegisterEventResponse	1.3.18.0.2.12.1
Event notification unregister request	UnregisterEventRequest UnregisterEventResponse	1.3.18.0.2.12.3
Get lines	ReadLogRequest ReadLogResponse	1.3.18.0.2.12.22 / 1.3.18.0.2.12.23

Table 5. Java classes for extended operations provided in IBM Security Directory Server JNDI Toolkit (continued)

Extended operations	Java classes	Request OID or Response OID
Get number of lines	GetLogSizeRequest GetLogSizeResponse	1.3.18.0.2.12.24 / 1.3.18.0.2.12.25
Group evaluation	EvaluateGroupsRequest EvaluateGroupsResponse	1.3.18.0.2.12.50 / 1.3.18.0.2.12.52
Kill connection	UnbindRequest UnbindResponse	1.3.18.0.2.12.35 / 1.3.18.0.2.12.36
LDAP trace facility	RemoteTraceExecutionRequest RemoteTraceExecutionResponse	1.3.18.0.2.12.41
LogMgmtControl	LogManagementRequest LogManagementresponse	1.3.18.0.2.12.70
Proxy back-end server resume role	ResumeRoleRequest ResumeRoleResponse	1.3.18.0.2.12.65
Quiesce or unquiesce replication context	QuiesceRequest QuiesceResponse	1.3.18.0.2.12.19
Replication error log	ControlReplErrorRequest ControlReplErrorResponse	1.3.18.0.2.12.56
Replication topology	ReplicationTopologyRequest ReplicationTopologyResponse	1.3.18.0.2.12.54 / 1.3.18.0.2.12.55
ServerBackupRestore	BackupRestoreRequest BackupRestoreResponse	1.3.18.0.2.12.81
Start, stop server	StartStopServerRequest StartStopServerResponse	1.3.18.0.2.12.26
Start TLS	StartTLSRequest StartTLSResponse	1.3.6.1.4.1.1466.20037
Unique attributes	UniqueAttributeRequest UniqueAttributeResponse	1.3.18.0.2.12.44 / 1.3.18.0.2.12.45
User type	UserTypeRequest UserTypeResponse	1.3.18.0.2.12.37 / 1.3.18.0.2.12.38

Implementing controls by using IBM Security Directory Server JNDI Toolkit

The LDAP V3 uses controls to send and receive more data to affect the behavior of predefined LDAP operations.

The controls are tagged along with LDAP operations and are sent to the server and are sent to as request controls. For example, a sort control can be sent with an LDAP search operation to request for the results be returned in a particular order. Solicited and unsolicited controls can also be returned along with responses from the server and are referred to as response controls. For example, an LDAP server might define a special control to return change or event notifications. All the supported controls in IBM Security Directory Server are also available in Java classes, in addition to C APIs. For a control, both the request control class and the response control class for controls that have response are available. For example, Limit number of attribute values on a Search Control has both request and response classes. The request and response control classes override the fields for ID, criticality, and the constructor for creation of control by using supplied arguments by application programs. The request control class also has setter methods specific to a control. It has constructors that allow the application to construct the control in all supported ways. A response control class has getter methods for the fields that can be retrieved for the control.

A class that represents a control extends the `javax.naming.ldap.BasicControl` class. This class defines following constructors and methods:

BasicControl(String ID)
: This constructor creates a noncritical control.

BasicControl(String ID, boolean criticality, byte[] value)
: This constructor creates a control by using the supplied arguments.

public String getID()
: This method retrieves the object identifier that is assigned for the LDAP control.

public boolean isCritical()
: This method determines the criticality of an LDAP control. IBM Security Directory Server must not ignore a critical control. If a server receives a critical control that it does not support, regardless of whether the control makes sense for the operation, the operation is not done. An `OperationNotSupportedException` exception is thrown. This method takes the value true as parameter if control is critical and false otherwise.

public byte[] getEncodedValue()
: This method retrieves the ASN.1 BER encoded value of an LDAP control. The result is raw BER bytes that include the tag and length of the control value. The result does not include the control OID or criticality. If the value is absent, NULL is returned. This method can be decoded by using `com.ibm.asn1.BERDecoder` in the IBM LDAP Java BER package.

An example of implementation control class:

```
public class DoNotReplicateControl extends javax.naming.ldap.BasicControl
{
      private static final String OID = "1.3.18.0.2.10.50";
      private byte[] berMess;
      private boolean criticality = false;

      public DoNotReplicateControl() {
```

```
                berMess = null;
        }

    public DoNotReplicateControl(boolean criticality) {
                this.riticality = criticality;
                berMess = null;
        }
}
```

The Java classes for controls that are provided in IBM Security Directory Server JNDI Toolkit are listed.

Table 6. Java classes for controls provided in IBM Security Directory Server JNDI Toolkit

Controls	Java classes	OID
Audit	AuditChainControl	1.3.18.0.2.10.22
Do not replicate	DoNotReplicateControl	1.3.18.0.2.10.23
Entry change notification	EntryChangeRequestControl EntryChangeResponseControl	2.16.840.1.113730.3.4.7
Group authorization	GroupAuthorizationControl	1.3.18.0.2.10.21
Limit number of attribute values	LimitAttributesSearchControl LimitAttributesSearchResponse Control	1.3.18.0.2.10.30
ibm-saslDigestBindRealmName	MD5RealmConnectionControl	1.3.18.0.2.10.12
ibm-saslDigestBindUserName	MD5UserConnectionControl	1.3.18.0.2.10.13
Manage DSAIT	ManageDSAITControl	2.16.840.1.113730.3.4.2
Modify groups only	ModifyGroupsOnlyControl	1.3.18.0.2.10.25
No replication conflict resolution	DoNotResoveReplication CoflictControl	1.3.18.0.2.10.27
Omit group referential integrity	OmitGroupReferential IntegrityControl	1.3.18.0.2.10.26
Paged search results	PagedResultsControl PagedResultsResponseControl	1.2.840.113556.1.4.319
Password policy request	PasswordPolicyRequestControl PasswordPolicyResponseControl	1.3.6.1.4.1.42.2.27.8.5.1/ 1.3.6.1.4.1.42.2.27.8.5.1
Persistent search	PersistentSearchControl	2.16.840.1.113730.3.4.3
Proxy authorization	ProxiedAuthorizationControl	2.16.840.1.113730.3.4.18
Return deleted objects	TombstoneControl	1.3.18.0.2.10.33
Server administration	ServerAdminControl	1.3.18.0.2.10.15
Sorted search results	SortedResultsControl SortedResultsResponseControl	1.2.840.113556.1.4.473
Subtree delete	TreeDeleteControl	1.2.840.113556.1.4.805
Transaction	TransactionControl	1.3.18.0.2.10.5

Table 6. Java classes for controls provided in IBM Security Directory Server JNDI Toolkit (continued)

Controls	Java classes	OID
Virtual list view	VLVRequestControl VLVResponseControl	2.16.840.1.113730.3.4.9/ 2.16.840.1.113730.3.4.10

LDAP client utilities

Currently, example source codes for some of the LDAP client utilities for basic LDAP operations like add, modify delete, search, and `modrdn` are provided both in C and Java, which can be used to build your own version of these LDAP client utilities.

The Java classes for each of these operations use JNDI APIs for doing the operations on directory server. The Java classes for the LDAP client utilities are provided in the *DS_INSTALL_ROOT*/`examples/java` directory.

Note: Javadoc HTML documentation for the extended operations and controls are zipped into `TDSJNDIToolkitJavaDocs.zip` and is available in the *DS_INSTALL_ROOT*/`javalib` directory.

The following LDAP client utilities in Java are available in the `examples/java` directory:

- **LDAPAdd** – To add LDAP entries to the server.
- **LDAPModify** – To modify LDAP entries on the server.
- **LDAPDelete** – To delete LDAP entries from the server.
- **LDAPModRDN** – To modify the RDN or DN of the entries on the server.
- **LDAPExop** – To run extended operations on the server.
- **LDAPSearch** – To search entries on the server.

The LDAP client utilities are compiled as Java class files. The table lists Java class files that are associated with their corresponding LDAP clients:

Table 7. LDAP clients and corresponding Java classes

LDAP clients	Java class file
LDAPAdd	`com.ibm.ldap.bp.client.ldapadd.LDAPAdd`
LDAPModify	`com.ibm.ldap.bp.client.ldapmodify.LDAPModify`
LDAPDelete	`com.ibm.ldap.bp.client.ldapdelete.LDAPDelete`
LDAPModRDN	`com.ibm.ldap.bp.client.ldapmodrdn.LDAPModRDN`
LDAPExop	`com.ibm.ldap.bp.client.ldapexop.LDAPExop`
LDAPSearch	`com.ibm.ldap.bp.client.ldapsearch.LDAPSearch`

To run the LDAP clients, you require IBM Java 5 or latter versions. For the LDAP clients to run, the class path must be correct and the path of the JAR files must be provided in the class path. The JAR files, `TDSJNDIToolkit.jar` and `IBMLDAPJavaBer.jar` are available in the *DS_INSTALL_ROOT*/`javalib` directory.

Note: To make Java clients to work over SSL and with key database files (with `kdb` extension), register a security provider for reading CMS in the `java.security` file.

To compile the source files to a wanted location, you can use the build script available in the *DS_INSTALL_ROOT*/examples/java directory. On Windows platform, the build script is provided as a batch file, build.bat, and on UNIX platforms, it is provided as a shell script, build.sh.

To run the script on UNIX platform, enter the following command at command prompt:

```
# ./build.sh
```

This script compiles all the LDAP Java clients to the bin folder. To run the script with parameters, check the usage instructions for the script by providing "-?" on Windows platform and "--help" on UNIX platform.

After the clients are compiled at the default location, *DS_INSTALL_ROOT/examples/java/bin*, the following command can be used to display the LDAPAdd client usage on a UNIX platform:

```
# pwd /opt/ibm/ldap/V6.3.1/examples/java/bin

# ../../../java/bin/java -classpath
.:../../../javalib/TDSJNDIToolkit.jar:../../../javalib/IBMLDAPJavaBer.jar com.\
                                    ibm.ldap.bp.client.ldapadd.LDAPAdd -?
```

Chapter 4. Change tracking in IBM Security Directory Server

You can track changes in the IBM Security Directory Server data for your requirements.

IBM Security Directory Server, version 6.1 and later versions provide different ways to track changes that are made to the directory data. Change tracking mechanism can be broadly categorized into notification that is based and poll based.

- In notification-based change-tracking mechanism, clients are notified about the changes to the directory data as and when they occur. Persistent search and event notification are the two notification-based change-tracking techniques.
- In poll based change tracking mechanism, clients are required to query the directory server for changes. LDAP clients can use the change log generated by IBM Security Directory Server to poll for changes.

Persistent search

Persistent search is an extended form of the standard LDAP search operation.

Persistent search sends the set of entries that match the search criteria. It also provides clients a means to receive change notifications to the LDAP server on entries within the result set that were sent to the client.

The persistent search control can be included in the Controls portion of an LDAP V3 search request message. The `controlType` for the persistent search control is `"2.16.840.1.113730.3.4.3"`.

```
PersistentSearch ::= SEQUENCE {
                       changeTypes INTEGER,
                       changesOnly BOOLEAN,
                       returnECs BOOLEAN
                     }
```

On receiving this control, IBM Security Directory Server processes the request as a standard LDAP V3 search with the following exceptions:

- If `changesOnly` is `TRUE`, the server does not return any existing entries that match the search criteria. Entries are only returned when they are changed by an update operation such as add, modify, delete, or `modifyDN` operation.
- After the changes are made to the server, the affected entries that match the search criteria are returned to the client only if the operation that caused the change is included in the **changeTypes** field. The **changeTypes** field is the logical or of one or more of these values: add (1), delete (2), modify (4), and `modDN` (8).
- After the operation is done, the server does not return a `SearchResultDone` message. Instead, the search operation is kept active until the client unbinds.
- If the value in the **returnECs** field is `TRUE`, the server returns the Entry Change Notification control with each entry returned as the result of changes.

The `ldap_create_persistentsearch_control()` API can be used to create the persistent search control that can then be passed to the controls section of the `ldap_search_ext()` or `ldap_search_ext_s()` API to initiate a persistent search.

The entry change notification control provides more information about the change that caused a particular entry to be returned on doing a persistent search. The `controlType` for the entry change notification control is `"2.16.840.1.113730.3.4.7"`.

If a client sets the **returnECs** field to `TRUE` in the persistent search control, then directory server includes the entry change notification control in the Controls portion of each `SerachResultEntry` that is returned due to an entry that is being added, deleted, or modified.

```
EntryChangeNotification ::= SEQUENCE {
           changeType ENUMERATED {
              add           (1),
              delete        (2),
              modify        (4),
              modDN         (8)
           },
           previousDN   LDAPDN  OPTIONAL,  # modifyDN operations only
           changeNumber INTEGER OPTIONAL  # if supported
}
```

where,

changeType
: This parameter indicates the type of LDAP operation that caused the entry to be returned.

previousDN
: The value of this parameter is present only for `modifyDN` operations. This parameter contains the DN of the entry before it was renamed or moved. The **returnECs** optional field is included only when you return change notifications as a result of `modifyDN` operations.

changeNumber
: This parameter contains the change number, `[CHANGELOG]`, assigned by the server for a change on an entry.

The `ldap_parse_entrychange_control()` API goes through a list of controls that are received from a persistent search operation, retrieves the entry change control from it and parses that control for change information.

See the `ldapsearch.c` example source code in the *`DS_INSTALL_ROOT`*`/examples` directory to know how to use persistent search.

Event notification

Event notifications are database objects that send information about server and database events to a client. It also provides the information that is required to understand, design, and implement event notifications.

The event notification function allows a server to notify a registered client that an entry in the directory tree is changed, added, or deleted. This notification is in the form of an unsolicited message.

Registration request

To register, the client must use a bound connection. To register a client, use the supported client APIs for extended operations. An LDAP v3 extended operation request has the form:

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

where the **`requestValue`** has the form:

```
requestValue ::= SEQUENCE {
                eventID         ENUMERATED {
                        LDAP_CHANGE (0)},
                baseObject      LDAPDN,
                scope           ENUMERATED {
                        baseObject              (0),
                        singleLevel             (1),
                        wholeSubtree            (2) },
                type    INTEGER OPTIONAL }
```

and where **type** has the form:

```
changeType ::= ENUMERATED {
                changeAdd               (1),
                changeDelete            (2),
                changeModify            (4),
                changeModDN             (8) }
```

Note: If the **type** field is not specified, it defaults to all changes.

An LDAP v3 extended operation response has the form:

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

Registration response

If the registration is successful, the server returns the following message and a unique registration ID:

```
LDAP_SUCCESS registration ID
```

If the registration fails, the server returns one of the following codes:

```
LDAP_UNWILLING_TO_PERFORM
```

This error code is returned if:

- The event notification function is turned off in the server.
- The event ID requested by the client cannot be handled by the server.
- The client is unbound.

```
LDAP_NO_SUCH_OBJECT
```

This error code is returned if:

- The base DN supplied by the client does not exist or is not visible to the client.

```
LDAP_NOT_SUPPORTED
```

This error code is returned if:

- The change type that is supplied by the client cannot be handled by the server.

Usage

When an event occurs, the server sends a message to the client as an LDAP v3 unsolicited notification. The message ID is 0 and the message is in the form of an extended operation response. The **responseName** field is set to the registration OID. The **response** field contains the unique registration ID and a timestamp for when the event occurred. The **time** field is in Coordinated Universal Time (UTC) format.

Note: When a transaction occurs, the event notifications for the transaction steps cannot be sent until the entire transaction is completed.

Unregistering a client

Set the **requestName** field to the unregister request OID. In the **requestValue** field type the unique registration ID returned by the server from the registration request:

```
requestValue ::= OCTET STRING
```

If the registration is successfully removed, the **LDAPResult** field contains `LDAP_SUCCESS` and the response field contains the registration ID that was removed.

If the unregistration request was unsuccessful, `NO_SUCH_OBJECT` is returned.

Example

```
#include stdio.h
#include string.h
#include ldap.h


struct berval *create_reg(int id,char *base,int scope,int type){
  struct berval *ret;
  BerElement *ber;

  if((ber = ber_alloc_t(1)) == NULL){
    printf("ber_alloc_t failed\n");
    return NULL;
  }
  if(ber_printf(ber,"{esi",id,base,scope) == (-1)){
    printf("first ber_printf failed\n");
    return NULL;
  }
  if(type != (-1)){
    if(ber_printf(ber,"i",type) == (-1)){
      printf("type ber_printf failed\n");
      return NULL;
    }
  }
  if(ber_printf(ber,"}") == (-1)){
    printf("closing ber_printf failed\n");
    return NULL;
  }

  if(ber_flatten(ber,&ret) == (-1)){
    printf("ber_flatten failed\n");
    return NULL;
  }
  ber_free(ber,1);
  return ret;
}

int main(int argc,char **argv){
  LDAP *ld;
  char *oidreq = "1.3.18.0.2.12.1";
  char *oidres;
  struct berval *valres = NULL;
  struct berval *registration;
  int rc,version, port;
  LDAPMessage *res;
  BerElement *ber;
  char *regID;

  argc--; argv++;

  port = 389;
  if(argc > 0){
    if(argc > 1) sscanf(argv[1],"%d",&port);
    ld = ldap_init(argv[0],port);
  }
  else
    ld = ldap_init("localhost",389);
  if(ld == NULL){
    printf("ldap_init failed\n");
    ldap_unbind(ld);
    return -1;
  }
  version = 3;
  ldap_set_option(ld,LDAP_OPT_PROTOCOL_VERSION,&version);

  if(ldap_simple_bind_s(ld,"cn=admin","secret") != LDAP_SUCCESS){
    printf("Couldn't bind\n");
    ldap_unbind(ld);
    return -1;
  }

  registration = create_reg(0,"o=sample",2,15);
  rc = ldap_extended_operation_s(ld,oidreq,registration,NULL,NULL,
                                 &oidres,&valres);
  if(rc == LDAP_SUCCESS){
    if(valres != NULL){
      if((ber = ber_init2(valres)) == NULL)
        printf("ber_init2 failed\n");
      else{
        if(ber_scanf(ber,"a",&regID) == LBER_ERROR)
          printf("ber_scanf failed\n");
        printf("registration ID: %s\n",regID);
        ber_free(ber,1);
      }
    }
    else{
      printf("valres NULL\n");
    }
  }
  else{
    printf("extended operation failed 0x%x\n",rc);
  }

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

Change log

The change log contains information about the IBM Security Directory Server operations.

IBM Security Directory Server records changes made to the LDAP data in the change log database. Entries in the change log database can be queried by using the standard LDAP APIs. All update operations to the directory server are recorded in this database.

LDAP client applications that depend on polling for identifying changes can query the change log periodically with appropriate filters (based on chronological change numbers).

All change log entries are of object class `ibm-changelog` and is derived from `changelogentry` object class. They are located under the DN entry `"cn=changelog"`. The following table contains attributes of the `ibm-changelog` object class with the description of a change.

Table 8. Attributes in the `ibm-changelog` object class and their description

Attribute	Description
changenumber	A number that uniquely identifies a change that is made to a directory entry. This integer value increases as new entries are added and is unique for a specific instance.
targetdn	DN of the entry that was added, deleted, or modified. For the **modrdn** operation, it gives the DN of the entry before it was modified.
changetype	Type of change that is made to the entry: add, modify, delete, or **modrdn**.
changes	The changes that are made to the directory server published in LDIF.
newRDN	The new RDN of an entry, if **changetype** is **modrdn**.
deleteOldRDN	It is a Boolean attribute. If the value is `TRUE`, it indicates that the RDN must not be retained as a distinguished attribute of the entry. If `FALSE`, it indicates that the RDN must be retained as a distinguished attribute.
newSuperior	If present, it gives the name of the immediate parent of the existing entry.
changetime	Time when the change was done.
ibm-changeInitiatorsName	DN of the user who initiated the change.

Chapter 5. LDAP client plug-in programming reference

The LDAP client plug-in programming reference provides information about writing client plug-ins.

Introduction to client SASL plug-ins

Client-side SASL plug-ins are used to extend the authentication capabilities of the LDAP client library.

The plug-ins work by intercepting the application invocation of the `ldap_sasl_bind_s()` API.

Note: SASL plug-ins are not designed to intercept asynchronous SASL binds.

Basic processing

The following section describes the typical flow when a SASL plug-in is used to provide an extended authentication function.

This flow assumes that the SASL plug-in shared library is already loaded by the LDAP library:

1. Application calls `ldap_sasl_bind_s()`, with a mechanism supported by a configured SASL plug-in.
2. The LDAP library calls the SASL bind worker function, as provided by the appropriate plug-in. The parameters that are supplied on the original `ldap_sasl_bind_s()` API are passed to the plug-in as elements of a `pblock` structure.
3. The plug-in worker function receives control, and extracts the parameters from the `pblock` by using the `ldap_plugin_pblock_get()` API. The following SASL-related information can be obtained from the `pblock` by the plug-in:
   - Distinguished Name (`dn`)
   - Credentials
   - Server controls
   - Client controls
   - Mechanism (plug-in subtype)

   In addition to these parameters, the plug-in can also obtain other information by using the `ldap_plugin_pblock_get()`, including:
   - Plug-in configuration information (that is, configuration information that is supplied in ARGC and ARGV form)
   - Target LDAP server host name
4. The plug-in runs its mechanism-specific logic. Some of the sample mechanisms that can be implemented as SASL plug-ins are as follows. They can be made available to all LDAP applications that are running on the system:

   Authentication that is based on a user fingerprint (for example, mechanism=userfp)

   When the fingerprint plug-in gets control, it uses the DN supplied on the `ldap_sasl_bind_s()` API to obtain an image of the user fingerprint. This authentication can entail prompting the user to use a fingerprint-scanning device. In this example, the fingerprint image, however obtained, represents the user credentials.

When the credentials are obtained, the plug-in is ready to run the actual SASL bind. This authentication is done by calling the `ldap_plugin_sasl_bind_s()` API, supplying the appropriate parameters (DN, credentials, mechanism, server controls). This API is a synchronous API that sends the SASL bind request to the LDAP server. Two items are returned to the plug-in when the bind result is returned from the server, and control is returned to the plug-in:

- Bind result error code
- Server credentials

If the server credentials are to be returned to the application, they must be set in the `pblock` before it returns control to the LDAP library, and after to the application. This setting is done by using `ldap_plugin_pblock_set()`. In this example, the plug-in work is complete, and it returns, supplying the bind result error code as the return code.

Authentication by using credentials that are previously established by the operating system

When the plug-in gets control, it queries the local security context to obtain the user identity and security token. For this example, we assume the user identity, as associated with the local security context, is used to construct the DN. The information from the security token is used for credentials.

After the credentials are obtained, the plug-in calls `ldap_plugin_sasl_bind_s()`, supplying the appropriate parameters (DN, credentials, mechanism, server controls). As in the previous example, the plug-in waits for the results of the bind request, then returns to the LDAP library, again setting server credentials in the `pblock`, if appropriate. Control is then returned to the application, along with the optional server credentials.

Authentication by using multiple binds (mechanism=DIGEST-MD5)

Some SASL mechanisms require multiple transactions between the client and the server (for example, the SASL DIGEST-MD5 mechanism). For this type of mechanism, when the plug-in gains control, it actually calls the `ldap_plugin_sasl_bind_s()` API multiple times. On each bind operation, the plug-in can supply DN, credentials, mechanism, and server controls, which are passed to the server. The LDAP server can return a result and server credentials back to the client. The plug-in can use this information to formulate another bind, again sent to the server by using `ldap_plugin_sasl_bind_s()`. When the multi-bind flow is complete, the plug-in returns control to the LDAP library with the result and optional server credentials.

Restrictions

The plug-in must not use any LDAP APIs, which accept `ld` as the input.

This input results in deadlock, since the `ld` is locked until the bind processing is complete.

Initializing a plug-in

You can initialize an SASL plug-in to work with LDAP client plug-in programming reference.

A typical LDAP SASL plug-in contains two entry points:
- An initialization routine
- A worker routine, which implements the authentication function

When an instance of an application uses a SASL plug-in for the first time, the LDAP library obtains the configuration information for the plug-in. The configuration information can come from `ibmldap.conf` or might be supplied explicitly by the application with the `ldap_register_plugin()` API.

When the configuration information is located, the LDAP library loads the plug-in shared library and call its initialization routine. By default, the name of the initialization routine for a plug-in is `ldap_plugin_init()`. A different entry point can be defined in `ibmldap.conf`, or supplied on the `ldap_plugin_register()` API if the plug-in is explicitly registered by the application.

The plug-in initialization routine is responsible for supplying the address of its worker routine entry point, which actually implements the authentication function. This initialization is done by using `ldap_plugin_pblock_set()` to define the address of the worker routine entry point in the `pblock`. For example, the following code segment depicts a typical initialization routine, where `authenticate_with_fingerprint` is the name of the routine that is provided by the plug-in to run a fingerprint-based authentication:

```
int ldap_plugin_init ( LDAP_Pblock      *pb )
{
        int rc;

        rc =  ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_BIND_S_FN, ( void * )
              authenticate_with_fingerprint );
        if ( rc != LDAP_SUCCESS ) printf("ldap_plugin_init couldn't initialize
              worker function\n");
        return ( rc );
}
```

A `pblock` is an opaque structure in which parameters are stored. A `pblock` is used to communicate between the LDAP client library and a plug-in. The `ldap_plugin_pblock_set` and `ldap_plugin_pblock_get` APIs are provided for your plug-in to set, or get, parameters in the `pblock` structure.

Using `ldap_plugin_pblock_get()`, the plug-in can also access configuration parameters. For example, the following code segment depicts how the plug-in can access its configuration information:

```
   int argc;
   char ** argv;

   rc = ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_ARGC, &argc );
   if (rc != LDAP_SUCCESS)
      return (rc);
   rc = ldap_plugin_pblock_get( pb, LDAP_PLUGIN_ARGV, &argv );
   if (rc != LDAP_SUCCESS)
      return (rc);
```

If the plug-in initialization processing is significant, and the results are to be preserved and made available to the plug-in worker function, the initialization routine can store the initialization results as private instance data in its shared library. When the plug-in worker function is later called, it can access this private instance data. For example, during initialization, the plug-in might be required to establish a session with a remote security server. Session information can be retained in the private instance data, which can be accessed later by the plug-in worker function.

After your plug-in is correctly initialized, its worker function can be used by the LDAP library. Continuing the example that is shown, if the mechanism supported by the plug-in is `userfp`, the `authenticate_with_fingerprint` function of your plug-in is called when the application issues an `ldap_sasl_bind_s()` function with `mechanism="userfp"`. See “Sample worker function” on page 175 for an example of a plug-in worker function.

Writing your own SASL plug-in

You can write your own SASL plug-in after you initialize an SASL plug-in.

About this task

Complete these steps to write your own SASL plug-in:

1. Implement your own initialization and worker functions. Include `ldap.h`, where you can find all the parameters that can be obtained from the `pblock`, and the function prototypes for the available plug-in functions:
   - `ldap_plugin_pblock_get()`
   - `ldap_plugin_pblock_set()`
   - `ldap_plugin_sasl_bind_s()`
2. Identify the input parameters to your initialization and worker functions.

   Note: The LDAP library can pass parameters to your plug-in initialization function by way of the argument list that is specified in `ibmldap.conf`, or by way of the **`plugin_parmlist`** parameter on the `ldap_register_plugin()` API. Information might also be supplied as client-side controls.
3. The initialization function must call the `ldap_plugin_pblock_set` API to register your plug-in worker function.
4. Implement your worker function. The worker function is responsible for obtaining the user credentials and implementing the authentication function. Typically this function involves calling the `ldap_plugin_sasl_bind_s()` API one or more times. If the authentication is successful, `LDAP_SUCCESS` must be returned. Otherwise, the unsuccessful LDAP result must be returned as the return code. If appropriate, the worker function can also return a value for server credentials.
5. Export your initialization function from your plug-in library. Use an `.exp` file for the AIX operating system or Solaris operating system, or a `.def` (or `dllexport`) file for the Windows NT operating system to export your initialization function.
6. Compile your client plug-in functions. Set the include path to include `ldap.h`, and to link to `ldap.lib`. Compile and link all your LDAP plug-in object files with whatever libraries you need, including `ldap.lib`. Make sure that the initialization function is exported from the `.dll` you created.
7. Add a plug-in directive in the LDAP plug-in configuration file, `ibmldap.conf`. Alternatively, the application can define the plug-in by calling the `ldap_register_plugin()` API.

Plug-in APIs

The following plug-in APIs illustrate the code formats.

For `pblock` access:

```
int ldap_plugin_pblock_get( LDAP_PBlock *pb, int arg, void **value );
int ldap_plugin_pblock_set( LDAP_PBlock *pb, int arg, void *value );
```

For sending an LDAP bind to the server:

```
int ldap_plugin_sasl_bind_s (
        LDAP            *ld,
        char            *dn,
        char            *mechanism,
        struct berval   *credentials,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        struct berval   **servercredp)
```

ldap_plugin_pblock_get()

The `ldap_plugin_pblock_get()` API returns the value that is associated with the specified `pblock` tag.

Syntax

```
#include "ldap.h"
int ldap_plugin_pblock_get( LDAP_PBlock *pb, int arg, void **value )
```

Parameters

pb Specifies the address of a `pblock`.

arg Specifies the tag or ID of the tag-value pair that you want to obtain from the `pblock`.

value Specifies a pointer to the address of the returned value.

Returns

Returns `0` if successful, or `-1` if an error occurs.

ldap_plugin_pblock_set()

The `ldap_plugin_pblock_set` API sets the value that is associated with the specified `pblock` tag.

Syntax

```
#include "ldap.h"
int ldap_plugin_pblock_set( LDAP_PBlock *pb, int arg, void *value );
```

Parameters

pb Specifies the address of a `pblock`.

arg Specifies the tag or ID of the tag-value pair that you want to set in the `pblock`.

value Specifies a pointer to the value that you want to set in the parameter block.

Returns

Returns `0` if successful, or `-1` if an error occurs.

ldap_plugin_sasl_bind_s()

The `ldap_plugin_sasl_bind_s` API is used by the plug-in to transmit an LDAP SASL bind operation to the LDAP server.

Syntax

```
#include "ldap.h"
int ldap_plugin_sasl_bind_s(
                    LDAP            *ld,
                    char            *dn,
                    char            *mechanism,
                    struct berval   *credentials,
                    LDAPControl     **serverctrls,
                    LDAPControl     **clientctrls,
                    struct berval   **servercredp)
```

Parameters

ld
Specifies the LDAP pointer that is associated with the application invocation of `ldap_sasl_bind_s()`. The plug-in obtains the `LD` with the `ldap_plugin_pblock_get()` API.

dn
Specifies the Distinguished Name to bind the entry. The DN might be supplied by the application and obtained by using `ldap_plugin_pblock_get()`, or it might be obtained by other means.

credentials
Specifies the credentials to authenticate with. Arbitrary credentials can be passed by using this parameter. The credentials might be supplied by the application and obtained by using `ldap_plugin_pblock_get()`, or they might be obtained by other means.

mechanism
Specifies the SASL mechanism to be used when it binds to the server. If a plug-in can be called for more than one mechanism, the plug-in can obtain the mechanism that was specified by the application with the `ldap_plugin_pblock_get()` API.

serverctrls
Specifies a list of LDAP server controls. For more information about server controls, see “LDAP controls” on page 27. The server controls might be supplied by the application and obtained by using `ldap_plugin_pblock_get()`, or they might be obtained by other means.

clientctrls
Specifies a list of LDAP client controls. For more information about client controls, see “LDAP controls” on page 27.

Note: The client controls are not supported currently for the `ldap_plugin_sasl_bind_s()` API.

Returns

error code
The error code is set to `LDAP_SUCCESS` if the bind succeeded. Otherwise, it is set to a nonzero error code.

servercredp
This result parameter is set to the credentials returned by the server. If no credentials are returned, it is set to `NULL`.

Sample worker function

An code example helps you to understand the sample worker function.

```
/*  Sample SASL Plug-in             */

#include ldap.h
#include string.h


int ldap_plugin_sasl_bind_s_prepare ( LDAP_Pblock   *pb )
{
        LDAP                    *ld;
        char                    *dn;
        char                    *mechanism;
        struct berval           *cred;
        LDAPControl             **serverctrls;
        LDAPControl             **clientctrls;
        struct berval           *servercredp = NULL;

        void *                  data;
        int                     rc;

        /***********************************************************************/
        /* Query pblock to obtain ld, dn, mechanism, credentials, server controls */
        /* and client controls, as supplied by application when it invoked the    */
        /* ldap_sasl_bind_s() API.                                                */
        /***********************************************************************/

        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_LD, &data ))){
                printf( "Could not get parameter for bind operation\n" );
                return ( rc );
        }
        ld = ( LDAP * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_DN,
          &data )))
                return ( rc );
        dn = ( char * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_MECHANISM,
          &data )))
                return ( rc );
        mechanism = ( char * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_CREDENTIALS,
          &data )))
                return ( rc );
        cred = ( struct berval * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_SERVERCTRLS,
          &data )))
                return ( rc );
        serverctrls = ( LDAPControl ** ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_CLIENTCTRLS,
          &data )))
                return ( rc );
        clientctrls = ( LDAPControl ** ) data;

        /***********************************************************************/
        /* Perform plug-in specific logic here to alter or obtain the user's      */
        /* distinguished name, credentials, etc.  This could include obtaining    */
        /* additional data from the pblock, including:                            */
        /*                                                                        */
        /*   LDAP_PLUGIN_TYPE       (e.g. "sasl")                                 */
        /*   LDAP_PLUGIN_ARGV       plug-in config variables                      */
        /*   LDAP_PLUGIN_ARGC       plug-in config variable count                 */
        /*                                                                        */
        /***********************************************************************/

        if ( rc = ( ldap_plugin_sasl_bind_s (
                                                ld,
                                                dn,
                                                mechanism,
                                                cred,
                                                serverctrls,
                                                clientctrls,
                                                &servercredp)))
                return rc;

        data = ( void * ) servercredp;

        if ( rc = ( ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_SERVER_CREDS,
          &data )))
                return rc;


        return ( LDAP_SUCCESS );
}


ldap_plugin_init ( LDAP_Pblock  *pb )
{
        int             argc;
        char            **argv;

        if ( rc = (ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_BIND_S_FN,
                                            ( void * )
          ldap_plugin_sasl_bind_s_prepare )))
            return ( rc );

        return ( LDAP_SUCCESS );
}
```

Appendix A. Possible extended error codes returned by LDAP SSL function codes

LDAP SSL function codes return possible extended error codes. The following information serves as a good starting point for the problems.

The following list contains values that are returned by all function calls:

- 0 – The task completed successfully. Issued by every function call that completes successfully.
- 1 – The environment or SSL handle is not valid. The specified handle was not the result of a successful open function call.
- 2 – The dynamic link library unloaded (Windows only).
- 3 – An internal error occurred. Report this error to service.
- 4 – Main memory is insufficient to run the operation.
- 5 – The handle is in an invalid state for operation, such as running an **init** operation on a handle twice.
- 6 – Specified key label not found in keyfile.
- 7 – Certificate not received from partner.
- 8 – Certificate validation error.
- 9 – Error processing cryptography.
- 10 – Error validating Abstract Syntax Notation (ASN) fields in certificate.
- 11 – Error connecting to LDAP server.
- 12 – Internal unknown error. Report problem to service.
- 101 – Internal unknown error. Report problem to service.
- 102 – I/O error reading keyfile.
- 103 – Keyfile has an invalid internal format. Re-create keyfile.
- 104 – Keyfile has two entries with the same key. Use **iKeyman** to remove the duplicate key.
- 105 – Keyfile has two entries with the same label. Use **iKeyman** to remove the duplicate label.
- 106 – The keyfile password is used as an integrity check. Either the keyfile is corrupted or the password ID is incorrect.
- 107 – The default key in the keyfile has an expired certificate. Use **iKeyman** to remove certificates that are expired.
- 108 – There was an error for loading one of the GSKdynamic link libraries. Be sure that GSK was installed correctly.
- 109 – Indicates that a connection is trying to be made in a gsk environment after the GSK_ENVIRONMENT_CLOSE_OPTIONS is set to GSK_DELAYED_ENVIRONMENT_CLOSE and gsk_environment_close() function is called.
- 201 – Neither the password nor the stash-file name was specified, so the key file could not be initialized.
- 202 – Unable to open the key file. Either the path was specified incorrectly or the file permissions did not allow the file to be opened.

- 203 – Unable to generate a temporary key pair. Report this error to service.
- 204 – A User Name object was specified that is not found.
- 205 – A Password that is used for an LDAP query is not correct.
- 206 – An index into the Fail Over list of LDAP servers was not correct.
- 301 – Indicates that the GSK environment close request was not properly handled. Cause is most likely because of a gsk_secure_socket*() command that is being attempted after a gsk_close_environment() call.
- 401 – The system date was set to an invalid value.
- 402 – Neither SSLv2 nor SSLv3 is enabled.
- 403 – The required certificate was not received from partner.
- 404 – The received certificate was formatted incorrectly.
- 405 – The received certificate type was not supported.
- 406 – An IO error occurred on a data read or write.
- 407 – The specified label in the key file could not be found.
- 408 – The specified key file password is incorrect. The key file could not be used. The key file might also be corrupted.
- 409 – In a restricted cryptography environment, the key size is too long to be supported.
- 410 – An incorrectly formatted SSL message was received from the partner.
- 411 – The message authentication code (MAC) was not successfully verified.
- 412 – Unsupported SSL protocol or unsupported certificate type.
- 413 – The received certificate contained an incorrect signature.
- 414 – Incorrectly formatted certificate received from partner.
- 415 – Invalid SSL protocol received from partner.
- 416 – Internal error. Report problem to service.
- 417 – The self-signed certificate is not valid.
- 418 – The read failed. Report this error to service.
- 419 – The write failed. Report this error to service.
- 420 – The partner closed the socket before the protocol completed.
- 421 – The specified V2 cipher is not valid.
- 422 – The specified V3 cipher is not valid.
- 423 – Internal error. Report problem to service.
- 424 – Internal error. Report problem to service.
- 425 – The handle could not be created. Report this internal error to service.
- 426 – Initialization failed. Report this internal error to service.
- 427 – When validating a certificate, unable to access the specified LDAP directory.
- 428 – The specified key did not contain a private key.
- 429 – A failed attempt was made to load the specified Public-Key Cryptography Standards (PKCS) #11 shared library.
- 430 – The PKCS #11 driver failed to find the token specified by the caller.
- 431 – A PKCS #11 token is not present in the slot.
- 432 – The password or pin to access the PKCS #11 token is invalid.

- 433 – The SSL header received was not a properly SSLV2 formatted header.
- 501 – The buffer size is negative or zero.
- 502 – Used with non-blocking input or output. See the non-blocking section for usage.
- 601 – SSLV3 is required for **reset_cipher**, and the connection uses SSLV2.
- 602 – An invalid ID was specified for the **gsk_secure_soc_misc** function call.
- 701 – The function call has an invalid ID. This may also be caused by specifying an environment handle when a handle for an SSL connection must be used.
- 702 – The attribute has a negative length, which is invalid.
- 703 – The enumeration value is invalid for the specified enumeration type.
- 704 – Invalid parameter list for replacing the SID cache routines.
- 705 – When setting a numeric attribute, the specified value is invalid for the specific attribute being set.
- 706 – Conflicting parameters have been set for additional certificate validation.

Appendix B. LDAP V3 schema

Use the following sections for information about the LDAP V3 schema.

Dynamic schema

IBM Security Directory Server, version 6.0 and later versions of C-Client SDK require that the schema defined for a server is stored in the `subschemasubentry` directory.

To access the schema, you must first determine the `subschemasubentry DN`, which is obtained by searching the root DSE. To obtain this information from the command-line, issue the following command:

```
ldapsearch -h hostname -p 389 -b "" -s base "objectclass=*"
```

The root DSE information that is returned from an LDAP V3 server, such as the IBM Directory server, includes the following subentry:

```
subschemasubentry=cn=schema
```

where `subschemasubentry DN` is `"cn=schema"`.

Using the `subschemasubentry DN` returned by searching the root DSE, schema information can be accessed with the following command-line search:

```
ldapsearch -h hostname -p 389 -b "cn=schema" -s base "objectclass=subschema"
```

The schema contains the following information:

Object class
A collection of attributes. A class can inherit attributes from one or more parent classes.

Attribute types
Contain information about the attribute, such as the name, `oid`, syntax, and matching rules.

IBM attribute types
The IBM LDAP directory implementation-specific attributes, such as database table name, column name, SQL type, and the maximum length of each attribute.

Syntaxes
Specific LDAP syntaxes available for attribute definitions.

Matching rules
Specific matching rules available for attribute definitions.

Schema queries

The **`ldapsearch`** utility can be used to query the subschema entry. This search can be run by any application by using the `ldap_search` APIs.

To retrieve all the values of one or more selected attribute types, specify the specific attributes that are wanted for the LDAP search. Schema-related attribute types include the following values:

- `objectclass`

- `objectclasses`
- `attributetypes`
- `ldapsyntaxes`
- `ibmattributetypes`
- `matchingrules`

For example, to retrieve all the values for `ldapsyntaxes`, specify:

```
ldapsearch -h host -b "cn=schema" -s base objectclass=* ldapsyntaxes
```

which returns something like:

```
cn=schema
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.10 DESC 'Certificate Pair' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.11 DESC 'Country String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.12 DESC 'DN' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.14 DESC 'Delivery Method' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.16 DESC 'DIT Content Rule
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.17 DESC 'DIT Structure Rule
        Description' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.21 DESC 'Enhanced Guide' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.22 DESC
       'Facsimile Telephone Number' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.23 DESC 'Fax' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.24 DESC 'Generalized Time' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.25 DESC 'Guide' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.26 DESC 'IA5 String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.27 DESC 'INTEGER' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.28 DESC 'JPEG' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.3 DESC 'Attribute Type
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.30 DESC 'Matching Rule
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.31 DESC 'Matching Rule Use
        Description' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.33 DESC 'MHS OR Address' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.34 DESC 'Name And Optional UID' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.35 DESC 'Name Form
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.36 DESC 'Numeric String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.37 DESC 'Object Class
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.38 DESC 'OID' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.39 DESC 'Other Mailbox' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.40 DESC 'Octet String' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.41 DESC 'Postal Address' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.42 DESC 'Protocol Information' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.43 DESC 'Presentation Address' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.44 DESC 'Printable String' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.49 DESC 'Supported Algorithm' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.5 DESC 'Binary' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.50 DESC 'Telephone
        Number' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.51 DESC
       'Teletex Terminal Identifier' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.52 DESC 'Telex Number' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.53 DESC 'UTC Time' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.54 DESC 'LDAP Syntax
        Description' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.58 DESC 'Substring Assertion' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.6 DESC 'Bit String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.7 DESC 'Boolean' )
```

```
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.8 DESC 'Certificate' )
ldapSyntaxes=( 1.3.6.1.4.1.1466.115.121.1.9 DESC 'Certificate List' )
ldapsyntaxes=( IBMAttributeType-desc-syntax-oid DESC 'IBM Attribute
        Type Description' )
```

Similarly, to obtain the values for matching rules, specify:

```
 ldapsearch -h host -b "cn=schema" -s base objectclass=* matchingrules
```

which returns something like:

```
cn=schema
     MatchingRules= ( 1.3.6.1.4.1.1466.109.114.3 NAME \
       'caseIgnoreIA5SubstringsMatch' SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
     MatchingRules= ( 2.5.13.5 NAME 'caseExactMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
     MatchingRules= ( 2.5.13.2 NAME 'caseIgnoreMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.7 NAME 'caseExactSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
     MatchingRules= ( 2.5.13.6 NAME 'caseExactOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.4 NAME 'caseIgnoreSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.58)
     MatchingRules= ( 2.5.13.3 NAME 'caseIgnoreOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 1.3.18.0.2.4.405 NAME 'distinguishedNameOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
     MatchingRules= ( 2.5.13.1 NAME 'distinguishedNameMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
     MatchingRules= ( 2.5.13.28 NAME 'generalizedTimeOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
     MatchingRules= ( 2.5.13.27 NAME 'generalizedTimeMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
     MatchingRules= ( 1.3.6.1.4.1.1466.109.114.2 NAME 'caseIgnoreIA5Match' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
     MatchingRules= ( 1.3.6.1.4.1.1466.109.114.1 NAME 'caseExactIA5Match' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
     MatchingRules= ( 2.5.13.29 NAME 'integerFirstComponentMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
     MatchingRules= ( 2.5.13.10 NAME 'numericStringSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
     MatchingRules= ( 2.5.13.11 NAME 'caseIgnoreListMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.41 )
     MatchingRules= ( 2.5.13.12 NAME 'caseIgnoreListSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
     MatchingRules= ( 2.5.13.13 NAME 'booleanMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
     MatchingRules= ( 2.5.13.14 NAME 'integerMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
     MatchingRules= ( 2.5.13.15 NAME 'integerOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
     MatchingRules= ( 2.5.13.16 NAME 'bitStringMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )
     MatchingRules= ( 2.5.13.17 NAME 'octetStringMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.5 )
     MatchingRules= ( 2.5.13.18 NAME 'octetStringOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )
     MatchingRules= ( 2.5.13.0 NAME 'objectIdentifierMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
     MatchingRules= ( 2.5.13.30 NAME 'objectIdentifierFirstComponentMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
     MatchingRules= ( 2.5.13.21 NAME 'telephoneNumberSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
     MatchingRules= ( 2.5.13.20 NAME 'telephoneNumberMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.50 )
     MatchingRules= ( 2.5.13.22 NAME 'presentationAddressMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.43 )
```

```
MatchingRules= ( 2.5.13.23 NAME 'uniqueMemberMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.34 )
MatchingRules= ( 2.5.13.24 NAME 'protocolInformationMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.42 )
MatchingRules= ( 2.5.13.25 NAME 'uTCTimeMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.53 )
MatchingRules= ( 2.5.13.8 NAME 'numericStringMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.36 )
MatchingRules= ( 2.5.13.9 NAME 'numericStringOrderingMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.36 )
```

Dynamic schema changes

To run a dynamic schema change, use LDAP modify with a DN of "cn=schema". It is permissible to add, delete, or replace only one schema entity, for example, an attribute type or an object class, at a time.

To delete a schema entity, you can provide the oid in parentheses:

```
( oid )
```

A full description might also be provided. In either case, the matching rule that is used to find the schema entity to delete is objectIdentifierFirstComponentMatch as mandated by the LDAP V3 protocol.

To add or replace a schema entity, you must provide the LDAP V3 definition and you can provide the IBM definition.

In all cases, you must provide only the definitions of the schema entity that you want to affect. For example, to delete the attribute type cn (its OID is 2.5.4.3), call ldap_modify() with:

```
LDAPMod  attr;
LDAPMod *attrs[] = { &attr, NULL };
char    *vals [] = { "( 2.5.4.3 )", NULL };
attr.mod_op      = LDAP_MOD_DELETE;
attr.mod_type    = "attributeTypes";
attr.mod_values  = vals;
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

To add an attribute type foo with OID 20.20.20 which is a NAME of length 20 chars:

```
char    *vals1[] = { "( 20.20.20 NAME 'foo' SUP NAME )", NULL };
char    *vals2[] = { "( 20.20.20 LENGTH 20 )", NULL };
LDAPMod  attr1;
LDAPMod  attr2;
LDAPMod *attrs[] = { &attr1, &attr2, NULL };
attr1.mod_op = LDAP_MOD_ADD;
attr1.mod_type = "attributeTypes";
attr1.mod_values = vals1;
attr2.mod_op = LDAP_MOD_ADD;
attr2.mod_type = "IBMattributeTypes";
attr2.mod_values = vals2;
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

To change the object class top so it allows a MAY attribute type called foo. It assumes that the attribute type foo is defined in the schema:

```
LDAPMod  attr;
LDAPMod *attrs[] = { &attr, NULL };
attr.mod_op = LDAP_MOD_REPLACE;
attr.mod_type = "objectClasses";
attr.mod_values = "( 2.5.6.0 NAME 'top' ABSTRACT "
                  "MUST objectClass MAY foo )";
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

Appendix C. LDAP distinguished names

Distinguished names (DNs) are used to uniquely identify entries in an LDAP or X.500 directory. DNs are user-oriented strings, typically used whenever you must add, modify, or delete an entry in a directory by using the LDAP programming interface, and when you use the LDAP utilities **`ldapmodify`**, **`ldapsearch`**, **`ldapmodrdn`**, and **`ldapdelete`**.

To know more about the syntax and usage of the command-line utilities, see *IBM Security Directory Server Command Reference*.

A DN is typically composed of an ordered set of attribute type or attribute value pairs. Most DNs are composed of pairs in the following order:
- `common name (cn)`
- `organization (o) or organizational unit (ou)`
- `country (c)`

The following string-type attributes represent the set of standardized attribute types for accessing an LDAP directory. A DN can be composed of attributes with an LDAP syntax of Directory String, including the following ones:
- `CN - CommonName`
- `L - LocalityName`
- `ST - StateOrProvinceName`
- `O - OrganizationName`
- `OU - OrganizationalUnitName`
- `C - CountryName`
- `STREET - StreetAddress`

Informal definition

This notation is convenient for common forms of name. Most DNs begin with `CommonName (CN)`, and progress up the naming tree of the directory. Typically, as you read from left to right, each component of the name represents increasingly larger groupings of entries, ending with `CountryName (C)`. Remember that sequence is important. For example, the following two DNs do not identify the same entry in the directory:

```
CN=wiley coyote, O=acme, O=anvils, C=US
```

```
CN=wiley coyote, O=anvils, O=acme, C=US
```

Some examples follow. The author of RFC 2253, "UTF-8 String Representation of Distinguished Names" is specified as:

```
CN=Steve Kille, O=ISODE Consortium, C=GB
```

Another name might be:

```
CN=Christian Huitema, O=INRIA, C=FR
```

A semicolon (;) can be used as an alternative separator. The separators might be mixed, but this usage is discouraged.

```
CN=Christian Huitema; O=INRIA; C=FR
```

Here is an example of a multi-valued Relative Distinguished Name, where the namespace is flat within an organization, and department is used to disambiguate certain names:

```
OU=Sales + CN=J. Smith, O=Widget Inc., C=US
```

The final examples show both methods of entering a comma in an Organization name:

```
CN=L. Eagle, O="Sue, Grabbit and Runn", C=GB
```

```
CN=L. Eagle, O=Sue, Grabbit and Runn, C=GB
```

Formal definition

For a formal, and complete, definition of Distinguished Names that can be used with the LDAP interfaces, see "RFC 2253, UTF-8 String Representation of Distinguished Names".

Appendix D. LDAP data interchange format (LDIF)

This documentation describes the LDAP Data Interchange Format (LDIF), as used by the **ldapmodify**, **ldapsearch**, and **ldapadd** utilities.

The LDIF specified here is also supported by the server utilities that are provided with the IBM Directory. To know more about the syntax and usage of the command-line utilities, see *IBM Security Directory Server Command Reference*.

LDIF is used to represent LDAP entries in text form. The basic form of an LDIF entry is:

```
dn: distinguished name
attrtype : attrvalue
attrtype : attrvalue
...
```

A line can be continued by starting the next line with a single space or tab character, for example:

```
dn: cn=John E Doe, o=University of High
 er Learning, c=US
```

Multiple attribute values are specified on separate lines, for example:

```
cn: John E Doe
cn: John Doe
```

If an *attrvalue* contains a non-US-ASCII character, or begins with a space or a colon (:), the *attrtype* is followed by a double colon and the value is encoded in base-64 notation. For example, the value `begins with a space` is encoded as:

```
cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=
```

Multiple entries within the same LDIF file are separated by a blank line. Multiple blank lines are considered a logical end-of-file.

LDIF examples

An LDIF content file contains entries that can be loaded to the directory.

Here is an example of an LDIF content file that contains three entries:

```
dn: cn=John E Doe, o=University of High
 er Learning, c=US
cn: John E Doe
cn: John Doe
objectclass: person
sn: Doe

dn: cn=Bjorn L Doe, o=University of High
 er Learning, c=US
cn: Bjorn L Doe
cn: Bjorn Doe
objectclass: person
sn: Doe

dn: cn=Jennifer K. Doe, o=University of High
 er Learning, c=US
cn: Jennifer K. Doe
cn: Jennifer Doe
```

```
objectclass: person
sn: Doe
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFx1Z2P/2wBDARESEhgVG
...
```

The `jpegPhoto` in the entry of Jennifer Doe is encoded by using base-64. The textual attribute values can also be specified in base-64 format. However, if so, the base-64 encoding must be in the code page of the wire format for the protocol. That is, for LDAP V2, the IA5 character set and for LDAP V3, the UTF-8 encoding.

LDIF example: Content

LDIF file: Change types

You can modify and delete existing directory entries when an LDIF file contains change types. For example, the following LDIF file entry shows the object class `insectopia` being added to the existing entry `dn= cn=foo, ou=bar` by using the modify change type:

```
dn: cn=foo, ou=bar
changetype: modify
add: objectclass
objectclass: insectopia
```

For a complete list of change types, see RFC 2849.

Change type files can also contain LDAP controls. LDAP controls can be used to extend certain LDAP Version 3 operations.

A control must contain a unique object identifier (OID) that identifies the control. Make sure that your server supports the control that you want to use.

The following example shows the LDAP control syntax. Brackets indicate optional data; only the OID is required.

```
control: OID [true||false] [string || :: 64string]
```

Where:
- *OID* is the OID that identifies the control you want to use.
- `string` is a string that does not include Line Feed, Carriage Return, NULL, colon, space or < symbol.
- *64string* is a base-64 encoded string.

The following example uses the Subtree delete control to delete the `ou=Product Development, dc=airius, dc=com` entry:

```
dn: ou=Product Development, dc=airius, dc=com
control: 1.2.840.113556.1.4.805 true
changetype: delete
```

When controls are included in an LDIF file, implementations might choose to ignore some or all of them. This implementation might be necessary if the changes described in the LDIF file are being sent on an `LDAPv2` connection (`LDAPv2` does not support controls), or the particular controls are not supported by the remote server. If the criticality of a control is "`true`", then the implementation must either include the control, or must not send the operation to a remote server.

For more information, see “LDAP controls” on page 27 and Appendix F, “Object Identifiers (OIDs) for extended operations and controls,” on page 197.

LDAP controls

Version 1 LDIF support

The **ldapmodify** and **ldapadd** client utilities are enhanced to recognize the latest version of LDIF, which is identified by the presence of the version: 1 tag at the head of the file.

Unlike the original version of LDIF, the newer version of LDIF supports attribute values that are represented in UTF-8, instead of the limited ASCII.

However, manual creation of an LDIF file that contains UTF-8 values can be difficult. To simplify this process, a `charset` extension to the LDIF format is supported. This extension allows an IANA character set name to be specified in the header of the LDIF file, along with the version number. A limited set of the IANA character sets is supported. See “IANA character sets supported by platform” on page 192 for the specific charset values that are supported for each operating system platform.

The version 1 LDIF format also supports file URLs. This format provides a more flexible way to define a file specification. File URLs take the following form:

```
attribute: file:///path
    (where path syntax depends on platform)
```

For example, the following addresses are valid file web addresses:

```
jpegphoto: file:///d:\temp\photos\myphoto.jpg
     (DOS/Windows style paths)
jpegphoto: file:///etc/temp/photos/myphoto.jpg
     (UNIX style paths)
```

Note: IBM Security Directory Server utilities support both the new file URL specification and the older style. For example, `jpegphoto: /etc/temp/myphoto`, regardless of the version specification. In other words, the new file URL format can be used without adding the version tag to your LDIF files.

Version 1 LDIF examples

You can use the optional `charset` tag so that the utilities automatically convert from the specified character set to UTF-8.

See the following example:

```
version: 1
charset: ISO-8859-1

dn: cn=Juan Griego, o=University of New Mexico, c=US
cn: Juan Griego
sn: Griego
description:: V2hhdCBhIGNhcmVmdWwgcmVhZGVyIHlvd
title: Associate Dean
title: [title in Spanish]
jpegPhoto: file:///usr/local/photos/jgriego.jpg
```

In this instance, all values that follow an attribute name and a single colon are translated from the ISO-8859-1 character set to UTF-8. Values following an attribute name and a double colon (such as `description:: V2hhdCBhIGNhcm...`) must be base-64 encoded, and are expected to be either binary or UTF-8 character strings. Values that are read from a file, such as the **jpegPhoto** attribute specified by the web address in the previous example, are also expected to be either binary or UTF-8. No translation from the specified `charset` to UTF-8 is done on those values.

In this example of an LDIF file without the `charset` tag, content is expected to be in UTF-8, or base-64 encoded UTF-8, or base-64 encoded binary data:

```
# IBM Directory sample LDIF file
#
# The suffix "o=sample" should be defined before attempting to load
# this data.

 version: 1

 dn: o=sample
 objectclass: top
 objectclass: organization
 o: sample

 dn: ou=Austin, o=sample
 ou: Austin
 objectclass: organizationalUnit
 seealso: cn=Linda Carlesberg, ou=Austin, o=sample
```

This same file can be used without the version: 1 header information, as in previous releases of the IBM Security Directory Server version C-Client SDK:

```
 # IBM Directory sample LDIF file
 #
 # The suffix "o=sample" should be defined before attempting to load
 # this data.

 dn: o=sample
 objectclass: top
 objectclass: organization
 o: sample

 dn: ou=Austin, o=sample
 ou: Austin
 objectclass: organizationalUnit
 seealso: cn=Linda Carlesberg, ou=Austin, o=sample
```

Note: The textual attribute values can be specified in base-64 format.

IANA character sets supported by platform

The following table defines the set of IANA-defined (Internet Assigned Numbers Authority) character sets that can be defined for the `charset` tag in a Version 1 LDIF file, on a per-platform basis.

The value in the left-most column defines the text string that can be assigned to the `charset` tag. An `X` indicates that conversion from the specified charset to UTF-8 is supported for the associated platform, and that all string content in the LDIF file is assumed to be represented in the specified charset. `n/a` indicates that the conversion is not supported for the associated platform.

String content is defined to be all attribute values that follow an attribute name and a single colon.

For more information about IANA-registered character sets, see IANA Character Sets.

Table 9. IANA-defined character sets by platform

Character	Conversion Supported				
Set Name	**Windows**	**AIX**	**Solaris**	**Linux**	**HP-UX**
ISO-8859–1	X	X	X	X	X
ISO-8859–2	X	X	X	X	X
ISO-8859–5	X	X	X	X	X
ISO-8859–6	X	X	X	X	X
ISO-8859–7	X	X	X	X	X
ISO-8859–8	X	X	X	X	X
ISO-8859–9	X	X	X	X	X
ISO-8859–15	NA	X	X		X
IBM437	X	NA	NA		NA
IBM850	X	X	NA		NA
IBM852	X	NA	NA		NA
IBM857	X	NA	NA		NA
IBM862	X	NA	NA		NA
IBM864	X	NA	NA		NA
IBM866	X	NA	NA		NA
IBM869	X	X	NA		NA
IBM1250	X	NA	NA		NA
IBM1251	X	NA	NA		NA
IBM1253	X	NA	NA		NA
IBM1254	X	NA	NA		NA
IBM1255	X	NA	NA		NA
IBM1256	X	NA	NA		NA
TIS-620	X	X	NA		NA
EUC-JP	NA	X	X	X	X
EUC-KR	NA	X	X*		NA
EUC-CN	NA	X	X		NA
EUC-TW	NA	X	X		X
Shift-JIS	X	X	X	X	NA
KSC	X	X	NA		NA
GBK	X	X	X*		NA
Big5	X	X	X		X
GB18030	X	X	X	X	NA
HP15CN					X (with non-GB18030)

* Supported on Solaris 7 and higher only.

The new Chinese character set standard (`GB18030`) is supported by appropriate patches available from http://www.oracle.com/us/sun/index.htm and http://www.microsoft.com/en-us/default.aspx:

Note: On Windows 2000, you must set the environment variable `zhCNGB18030=TRUE`.

Appendix E. Deprecated LDAP APIs

Although the following APIs are still supported, their use is deprecated.

Use of the newer replacement APIs is encouraged:

`ldap_ssl_start()`
: Use `ldap_ssl_client_init()` and `ldap_ssl_init()`. See "LDAP_SSL" on page 129.

`ldap_open()`
: Use `ldap_init()`. See"LDAP_INIT" on page 63.

`ldap_bind()`
: Use `ldap_simple_bind()`. See "LDAP_BIND / UNBIND" on page 10.

`ldap_bind_s()`
: Use `ldap_simple_bind_s()`. See "LDAP_BIND / UNBIND" on page 10.

`ldap_result2error()`
: Use `ldap_parse_result()`. See "LDAP_PARSE_RESULT" on page 92.

`ldap_perror()`
: Use `ldap_parse_result()`. See "LDAP_PARSE_RESULT" on page 92.

`ldap_get_entry_controls_np`
: Use `ldap_get_entry_controls`. See "LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE" on page 54.

`ldap_parse_reference_np`
: Use `ldap_parse_reference`. See "LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE" on page 54.

Appendix F. Object Identifiers (OIDs) for extended operations and controls

The extended operation and control OIDs are in the root DSE of IBM Security Directory Server.

In this appendix, each OID is defined and its syntax that is specified in the following formats:

Extended operations:

Description
Gives a brief description of the extended operation.

Request
OID and syntax for the extended operation request. A request generally sets the **requestValue** field.

Response
OID and syntax for the extended operation response.

Behavior
How the extended operation behaves; who is enabled to send the extended operation; possible return codes.

Scope The scope of the extended operation.

Auditing (if applicable)
How this extended operation is audited.

Controls:

Description
Gives a brief description of the control.

OID OID for the extended operation.

Syntax
Syntax for the control.

Behavior
How the control behaves; who is enabled to call the control; possible return codes.

Scope The scope of the control.

Auditing (if applicable)
How this control is audited.

OIDs for extended operations

The OIDs for extended operations provide support description about various servers.

The following table shows OIDs for extended operations. Click a short name or go to the specified page number for more information about an extended operation syntax and usage.

Table 10. OIDs for extended operations

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Account status extended operation" on page 202 1.3.18.0.2.12.58	This extended operation sends the server a DN of an entry which contains a **`userPassword`** attribute, and the server sends back the status of the user account that is being queried: open locked expired	No	Yes	No	No
"Attribute type extended operations" on page 203 1.3.18.0.2.12.46	Retrieve attributes by supported capability: operational, language tag, attribute cache, unique, or configuration. **Note:** From IBM Security Directory Server, version 6.3, attribute cache is deprecated. You must avoid using attribute cache.	Yes	Yes	Yes	Yes
"Begin transaction extended operation" on page 206 1.3.18.0.2.12.5	Begin a Transactional context.	No	Yes	Yes	Yes
"Cascading replication operation extended operation" on page 207 1.3.18.0.2.12.15	This operation calls the requested action on the server it is issued to and cascades the call to all consumers beneath it in the replication topology.	No	Yes	No	No
"Clear log extended operation" on page 243 1.3.18.0.2.12.20	Request to clear log file.	No	Yes	Yes	Yes
"Control replication extended operation" on page 210 1.3.18.0.2.12.16	This operation is used to force immediate replication, suspend replication, or resume replication by a supplier. This operation is allowed only when the client owns update authority to the replication agreement.	No	Yes	No	No

Table 10. OIDs for extended operations (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Control queue extended operation" on page 212 1.3.18.0.2.12.17	This operation marks items as "already replicated" for a specified agreement. This operation is allowed only when the client owns update authority to the replication agreement.	No	Yes	No	No
"DN normalization extended operation" on page 213 1.3.18.0.2.12.30	Request to normalize a DN or a sequence of DNs.	Yes	Yes	No	No
"Dynamic server trace extended operation" on page 214 1.3.18.0.2.12.40	Activate or deactivate tracing in the IBM Security Directory Server.	No	Yes	Yes	Yes
"Dynamic update requests extended operation" on page 216 1.3.18.0.2.12.28	Request to update server configuration for IBM Security Directory Server.	Yes	Yes	Yes	Yes
"Effective password policy extended operation" on page 217 1.3.18.0.2.12.75	Used for querying effective password policy for a user or a group.	No	Yes	No	No
"End transaction extended operation" on page 218 1.3.18.0.2.12.6	End Transactional context (commit or rollback).	No	Yes	Yes	Yes
"Event notification register request extended operation" on page 220 1.3.18.0.2.12.1	Request registration for events notification.	No	Yes	No	No
"Event notification unregister request extended operation" on page 221 1.3.18.0.2.12.3	Unregister for events that were registered for using an Event Registration Request.	No	Yes	No	No
"Get file extended operation" on page 244 1.3.18.0.2.12.73	Returns the contents of a file on the server.	No	Yes	Yes	Yes

Table 10. OIDs for extended operations (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Get lines extended operation" on page 245 1.3.18.0.2.12.22	Request to get lines from a log file.	Yes	Yes	Yes	Yes
"Get number of lines extended operation" on page 246 1.3.18.0.2.12.24	Request number of lines in a log file.	Yes	Yes	Yes	Yes
"Group evaluation extended operation" on page 221 1.3.18.0.2.12.50	Requests all the groups that a user belongs to.	No	Yes	No	No
"Kill connection extended operation" on page 223 1.3.18.0.2.12.35	Request to kill connections on the server. The request can be to kill all connections or kill connections by bound DN, IP, or a bound DN from a particular IP.	No	Yes	Yes	Yes
"LDAP trace facility extended operation" on page 224 1.3.18.0.2.12.41	Use this extended operation to control LDAP Trace Facility remotely by using the Administration Server.	Yes	Yes	Yes	Yes
"Locate entry extended operation" on page 225 1.3.18.0.2.12.71	This extended operation is used to extract the back-end server details of a set of entry DNs and provide the details to the client.	No	No	Yes	Yes
"LogMgmtControl extended operation" on page 226 1.3.18.0.2.12.70	The `LogMgmtControl` extended operation is used to start, stop, and query the status of the log management for a directory server instance that is running on a server.	Yes	Yes	Yes	Yes
"Online backup extended operation" on page 227 1.3.18.0.2.12.74	Runs online backup of the directory server instance DB2 database.	No	Yes	No	No

Table 10. OIDs for extended operations (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Password policy bind initialize and verify extended operation" on page 228 1.3.18.0.2.12.79	The Password policy-bind initialize and verify extended operation runs password policy bind initialization and verification for a specified user.	No	Yes	No	No
"Password policy finalize and verify bind extended operation" on page 229 1.3.18.0.2.12.80	The Password policy-finalize and verify bind extended operation runs password policy post-bind processing for a specified user.	No	Yes	No	No
"Prepare transaction extended operation" on page 230 1.3.18.0.2.12.64	Using the prepare transaction extended operation the client requests the server to start processing the operations that are sent in a transaction.	No	Yes	Yes	Yes
"Proxy back-end server resume role extended operation" on page 231 1.3.18.0.2.12.65	This extended operation enables a proxy server to resume the configured role of a back-end server in a distributed directory environment.	No	No	Yes	Yes
"Quiesce or unquiesce replication context extended operation" on page 233 1.3.18.0.2.12.19	This operation puts the subtree into a state where it does not accept client updates (or terminates this state). Only the updates from clients are authenticated as directory administrators where the Server Administration control is present.	No	Yes	No	No
"Replication error log extended operation" on page 234 1.3.18.0.2.12.56	Maintenance of a replication error log.	No	Yes	No	No
"Replication topology extended operation" on page 235 1.3.18.0.2.12.54	Trigger a replication of replication topology-related entries under a replication context.	No	Yes	No	No

Table 10. OIDs for extended operations (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"ServerBackupRestore extended operation" on page 236 1.3.18.0.2.12.81	Issues request to the administration server to do the following actions: • Back up a directory server data and configuration files • Restore directory server data and configuration from an existing backup	Yes	Yes	No	No
"Start, stop server extended operations" on page 238 1.3.18.0.2.12.26	Request to start, stop, or restart an LDAP server.	Yes	Yes	Yes	Yes
"Start TLS extended operation" on page 239 1.3.6.1.4.1.1466.20037	Request to start Transport Layer Security.	Yes	Yes	Yes	Yes
"Unique attributes extended operation" on page 240 1.3.18.0.2.12.44	The unique attributes extended operation provides a list of all non-unique (duplicate) values for a particular attribute.	No	Yes	No	No
"User type extended operation" on page 241 1.3.18.0.2.12.37	Request to get the User Type of the bound user.	Yes	Yes	Yes	Yes

Account status extended operation

The account status extended operation explains its use with the server and provides the results.

Description

This extended operation sends the server a DN of an entry which contains a **`userPassword`** attribute, and the server sends back the status of the user account that is being queried:

- open
- locked
- expired

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.58

Syntax

```
SEQUENCE {
    dn    LDAPDN
}
```

Response

OID 1.3.18.0.2.12.59

Syntax

```
SEQUENCE {
   status    INTEGER{open(0), locked(1), expired(2)};
}
```

Behavior

This extended operation requests the account status of a user account. The DN is the DN of the user account that is being queried. The server sends back the status of the user account that is being queried.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` and `PasswordAdmin` roles
- Global Administration Group members

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_NO_RESULTS_RETURNED`
- `LDAP_PROTOCAL_ERROR`

This extended operation is not supported by the Administration Server.

Scope The extended operation affects only the current operation.

Attribute type extended operations

The Attribute type extended operations explain their use with the server and provides the results.

Description

The server provides a way for LDAP clients to determine type of attributes in the schema. This extended operation is used to list attributes that have a specific characteristic. The extended operation also provides a way for LDAP clients to query about the following attributes:

- Operational - The operational attributes of the server.
- Language Tag - The attributes that can use language tags.
- Attribute Cache - The attributes that can be cached in attribute cache.

  Note: From IBM Security Directory Server, version 6.3, attribute cache is deprecated. You must avoid using attribute cache.
- Unique - The attributes that can be marked as unique.
- Configuration - The configuration attributes of the server.
- OS400 - The attributes that are used by the i5/OS™ system projection back-end (i5/OS V5R4).

- Encryptable - The attributes that can be defined as encryptable (version 6.1 and later).
- Encrypted - The attributes that are currently defined as encrypted in the server schema (version 6.1 and later). This returns a subset of encryptable attributes that might have any of the encryption-related settings: `ENCRYPT`, `RETURN-VALUE`, `SECURE-CONNECTION-REQUIRED` or `NONMATCHABLE`.

Request

OID 1.3.18.0.2.12.46

Syntax

```
RequestValue ::= SEQUENCE {
  AttributeTypeRequest  ENUMERATED {
        OPERATIONAL      (0),
        LANGUAGE TAG     (1),
        ATTRIBUTE CACHE  (2),
        UNIQUE           (3),
        CONFIGURATION     (4),
        OS400            (5), #i5/OS V5R4 or later
        ENCRYPTABLE      (6), # v6.1 or later
        ENCRYPTED        (7)  # v6.1 or later
  },
hasCharacteristic        BOOLEAN }
```

The extended operation request value takes two parameters on the request. The first parameter is an enumeration that tells the server that the attribute type (characteristic) is being requested. The extended operation supports queries for the following attributes:

- Operational
- Language Tag
- Attribute Cache

  Note: From IBM Security Directory Server, version 6.3, attribute cache is deprecated. You must avoid using attribute cache.
- Unique
- OS400
- Encryptable
- Encrypted

The second parameter is a Boolean value that determines whether to return the attributes that have the specified attribute characteristic. A value of FALSE returns a list of attribute names that do not fall into the specified attribute category. A value of `TRUE` returns a list of attribute names that do fall into the specified attribute category.

Response

OID 1.3.18.0.2.12.47

Syntax

```
ResponseValue ::= SEQUENCE of AttributeNames; #LDAPString or OCTET STRING
```

Result codes

A standard LDAP result code is returned in the **`resultCode`** component of the extended response message.

Note: If the result code is LDAP_SUCCESS, a list of the attributes that match the request criteria is returned in the response value.

Behavior

This extended operation enables the user to do the following actions:

- Retrieves a list of all operational attributes.
- Retrieves a list of all attributes that can use language tags (not a list of attributes that are using language tags).
- Retrieves a list of all attributes that can be cached (not a list of attributes that are being cached).
- Retrieves a list of all attributes that can be made unique attributes (not a list of attributes that are currently unique attributes).
- Allows the user to retrieve a list of all attributes that are configuration attributes. These attributes are defined in the configuration schema.
- Retrieves a list of attributes that are used by the i5/OS system projection.
- Retrieves a list of attributes that can be defined as encryptable (version 6.1 and later).
- Retrieves a list of attributes that are currently defined as encrypted in the server schema (version 6.1 and later).

This extended operation also provides an option to return the inverse of any attribute characteristic for which the user queries. For example, the user must be able to ask for all attributes that are not operational attributes.

If the encryption setting of a schema attribute type definition is changed, it is audited as a new audit event, AU_EVENT_ATTR_ENCRYPTION_CHANGED. The audit event message string is:

```
"GLPSCH045I Encryption setting for attribute '%1$s'
changed to ENCRYPT=%2$s SECURE-CONNECTION-ONLY=%3$s RETURN-VALUE=%4$s\n"
```

The **ENCRYPT** value is none or the specified scheme. The **SECURE-CONNECTION-ONLY** value can be either 'true' or 'false'. The RETURN-VALUE value can be cleartext or the specified scheme.

All user types, including anonymous users, are enabled to call this extended operation.This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OTHER
- LDAP_PROTOCAL_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS

This extended operation is supported by the Administration Server.

The authorization that is required for using this extended operation depends on the attribute type requested. The attribute type and the authority that is required are listed in the table.

Table 11. Authorization required for attributes

Attribute Type	Authority Required
Operational	Anonymous

Table 11. Authorization required for attributes (continued)

Attribute Type	Authority Required
Language Tag	Anonymous
Attribute Cache	Anonymous
Unique	Anonymous
Configuration	Anonymous
OS400	Anonymous
Encryptable	Primary Directory Administrator or Local Administration Group members with `DirDataAdmin` and `SchemaAdmin` roles
Encrypted	Primary Directory Administrator or Local Administration Group members with `DirDataAdmin` and `SchemaAdmin` roles

Scope This extended operation has no affect on subsequent requests.

Auditing

The Attribute Type extended operation has an audit string of the following form:

```
AttributeType: Type
```

where *Type* is one of the following types:

- Operational
- Language Tag
- Attribute Cache

  Note: From IBM Security Directory Server, version 6.3, attribute cache is deprecated. You must avoid using attribute cache.
- Unique Attribute
- Configuration
- OS400
- Encryptable
- Encrypted

```
hasCharacteristic: Boolean
```

where *Boolean* is one of the following values:

- FALSE
- TRUE

Begin transaction extended operation

The begin transaction extended operation explains its use with the server and provides the results.

Description

The Begin transaction extended operation sends requests to the server to start a transaction context on the connection.

Note: This extended operation is enabled by default, but can be disabled by changing the value of the **`ibm-slapdTransactionEnable`** attribute in the configuration file.

The **`ibm-slapdTransactionEnable`** attribute is in the `cn=Transaction, cn=Configuration` entry in the configuration file. If the value of this attribute is set to `FALSE`, transactions are disabled. If the value is set to `TRUE`, transactions are enabled. Transactions can also be enabled or disabled by using the web administration tool.

Request

OID 1.3.18.0.2.12.5

Syntax
There is no request value.

Response

OID 1.3.18.0.2.12.5

Syntax
The response value is a string that contains the transaction ID. The transaction ID is not BER encoded.

Note: A transaction ID is a string value that is generated by the directory server in response to a start transaction request.

Behavior
This extended operation puts the connection in the transaction state.

All users can run this extended operation.

This extended operation has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`

This extended operation is not supported by the Administration Server.

Scope This extended operation changes the state of the connection for future operations. This connection remains in the transaction state until a stop transaction extended operation is sent, or an error occurs.

Cascading replication operation extended operation

The cascading replication operation extended operation explains its use with the server and provides the results.

Description
Run a replication extended operation on every server in the full replication topology. This extended operation runs the requested action on the server on which it is issued. It cascades the call to all consumers beneath it in a replication topology.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.15

Syntax

```
requestValue ::= SEQUENCE {
action ActionValue,
subtreeDN DistinguishedName,
timeout INTEGER
}
ActionValue ::= INTEGER {
quiesce (0),
unquiesce (1),
replicateNow (2),
waitForReplication (3)
}
```

Response

OID 1.3.18.0.2.12.15

Syntax

```
responseValue ::= SEQUENCE {
    # LDAPResult fields
  resultCode INTEGER (0..MAX),
    errorMessage LDAPString

  # Operation specific failure information:
    supplier LDAPString,
    consumer LDAPString,

    # Additional optional fields:
    additionalResultCode [1] INTEGER OPTIONAL,
    agreementDN [2] LDAPString OPTIONAL

}
```

When the **`resultCode`** is `LDAP_TIMEOUT`, the **additionalResultCode** field must be set to one of the following values:

```
  additionalResultCode ENUMERATED {
       LDAP_REPLICATION_SUSPENDED      [1],
       LDAP_REPLICATION_RETRYING       [2],
       LDAP_REPLICATION_ERROR_LOG_FULL [3]
    }
```

Note: The **additionalResultCode** and **agreementDN** fields are not present for servers earlier than IBM Security Directory Server, version 6.1.

The following codes are possible return codes:

- `LDAP_SUCCESS` - Operation was successful
- `LDAP_NO_SUCH_OBJECT` - Replication context or agreement does not exist
- `LDAP_UNWILLING_TO_PERFORM` - Object is not a replication context
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_INSUFFICIENT_ACCESS` - Not authorized to run the operation
- `LDAP_PARAM_ERROR`
- `LDAP_ENCODING_ERROR`
- `LDAP_LOCAL_ERROR`
- `LDAP_TIMEOUT` - Operation did not complete within specified time

Behavior

The requested operation is run on the target server and on all replicas of the target server. This extended operation runs the requested action on the server it is issued on. It cascades the call to all consumers beneath it in a replication topology. The operation returns when one of the following conditions occurs:

- The request is completed on all servers.
- A failure occurred on a server (result indicates the failure and the server).
- The timeout value is exceeded.

This extended operation is allowed only when:

- The client is authenticated with update authority to all agreements in the specified subtree.
- The client is authenticated as a master server for the specified subtree.

Sometimes when a “wait for replication” is called during the add replica, add master, or move operation in a replication, wait for replication time-out. No error is displayed that resulted in time-out. This error is occurred because the cascaded replication times out. To facilitate a better diagnosis, the replication response structure is updated. When the return code is `LDAP_TIMEOUT`, the **additionalResultCode** and **agreementDN** fields are set.

The **additionalResultCode** field is populated with error message. Following examples illustrate how the server handles the cascaded replication timeout cases and the possible error messages:

- `resultCode = LDAP_TIMEOUT` without **additionalResultCode** means a directory server instance earlier than 6.1.

  Web Administration tool and `ldapexop` displays a message, for example:
  ```
  Replication from supplier replica Supreplica_1 to consumer replica
   hostname: port did not complete.

  Replication agreement xxx is suspended.
  ```
- `resultCode = LDAP_TIMEOUT` with `additionalResultCode = LDAP_REPLICATION_SUSPENDED`

  Web Administration tool and `ldapexop` displays a message, for example:
  ```
  Replication from supplier replica Supreplica_1 to consumer replica
   hostname: port did not complete.

  Replication agreement xxx is suspended.
  ```
- `resultCode = LDAP_TIMEOUT` with `additionalResultCode = LDAP_REPLICATION_RETRYING`

  Web Administration tool and `ldapexop` displays a message, for example:
  ```
  Replication from supplier replica Supreplica_1 to consumer replica
   hostname: port did not complete.

  Replication agreement xxx is blocked on a failing change.
  ```
- `resultCode = LDAP_TIMEOUT` with `additionalResultCode = LDAP_REPLICATION_ERROR_LOG_FULL`

  Web Administration tool and `ldapexop` displays a message, for example:
  ```
  Replication from supplier replica Supreplica_1 to consumer replica
   hostname: port did not complete.

  The replication error log is full for agreement xxx.
  ```

The **agreementDN** field contains the DN of the associated replication agreement. The **agreementDN** field is set whenever the server that detects the error is working with a particular agreement.

This response is sent for all requests from servers that have a well-formed request value. The response value consists of a `resultCode` with `errorMessage` and information about where the error was detected.

The supplier field contains the DNS host name of the server that reports the error. If the error occurs with a consumer server, the consumer field contains the DNS host name of the consumer server. The error is the server that is timed out and waiting for a response from a consumer. In this case, the supplier field is always completed but the consumer field might be empty. Since it is an error condition, the **agreementDN** field is populated, which provides information about the supplier and consumer.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` and `ReplicationAdmin` roles
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_NO_MEMORY`
- `LDAP_DECODING_ERROR`
- `LDAP_UNDEFINED_TYPE`
- `LDAP_INVALID_DN_SYNTAX`

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

```
Action: [Quiesce | Unquiesce | ReplNow | Wait | Unknown]
Context DN: context DN
Timeout: timeout
```

Control replication extended operation

The control replication extended operation explains its use with the server and provides the results.

Description

This extended operation is used to control the following aspects of currently running replications:

- Suspend replication
- Resume replication
- Cause changes to be replicated immediately

Request

OID 1.3.18.0.2.12.16

Syntax

```
requestValue ::= SEQUENCE {
action ActionValue,
scope ScopeValue
entryDN DistinguishedName
}
ActionValue ::= INTEGER {
```

```
suspend (0),
resume (1),
replicateNow (2),
terminateFullReplication (3)
}
ScopeValue ::= INTEGER {
singleAgreement (0),
allAgreements (1)
}
```

Response

OID 1.3.18.0.2.12.16

Syntax

```
Response Value ::= SEQUENCE {
#fields of interest from LDAPResult:
resultCode INTEGER (0..MAX),
errorMessage LDAPString,
consumer LDAPString
}
```

The following return codes are possible:

- `LDAP_SUCCESS`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_NO_MEMORY`

Behavior

This extended operation is used to control the following aspects of currently running replications:

Suspend replication
Changes are not replicated for the replication agreement or for all replication agreements for the context until the resume replication or replicate immediately operation is used.

Resume replication
If the replication agreement is suspended, then replication resumes.

Cause changes to be replicated immediately
If the replication agreement is suspended or is waiting for scheduled replication to occur, any outstanding changes are replicated.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` and `ReplicationAdmin` roles
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`

- LDAP_PROTOCOL_ERROR
- LDAP_DECODING_ERROR
- LDAP_NO_MEMORY
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

```
Action: [Suspend | Resume | ReplNow | Unknown]
Scope: [Single | All | Unknown]
DN: dn
```

Control queue extended operation

The control queue extended operation explains its use with the server and provides the results.

Description
This extended operation is used to skip changes in the replication queue for an agreement.

Request

OID 1.3.18.0.2.12.17

Syntax

```
requestValue ::= SEQUENCE {
action ActionValue,
agreementDN DistinguishedName,
changeId LDAPString
}
ActionValue ::= INTEGER {
skipAll (0),
skipSingle (1)
}
```

Response

OID 1.3.18.0.2.12.17

Syntax

```
Response Value ::= SEQUENCE {
#fields of interest from LDAPResult:
resultCode INTEGER (0..MAX),
errorMessage LDAPString,
#operation information:
changesSkipped INTEGER (0..MAX)
}
```

The following codes are possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY

Behavior
This extended operation skips changes in the replication agreements queue.

If skipSingle is used, and changeID is the next ID in the replication agreements queue, then changeID is skipped over. If changeID is not at the head of the list of pending changes, the operation fails. If skipAll is used, then all outstanding changes in the replication agreements queue are skipped.

The following persons are enabled to call the extended operation:
- Primary Directory Administrator
- Local Administration Group members with DirDataAdmin and ReplicationAdmin roles
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:
- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_DECODING_ERROR
- LDAP_NO_MEMORY
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

```
Skip: [ All | changeId | Unknown ]
Agreement DN: agreementDn
```

DN normalization extended operation

The DN normalization extended operation explains its use with the server and provides the results.

Description

The DN normalization extended operation normalizes a DN or a list of DNs. The normalization is based on the server schema.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.30

Syntax

```
RequestValue ::= SEQUENCE {
     case   INTEGER {preserve(0), normalize (1)};
     SEQUENCE of DistinguishedName;
 }
```

Response

OID 1.3.18.0.2.12.30

Syntax

```
ResultValue ::= SEQUENCE {
  SEQUENCE of SEQUENCE {
  Return code  INTEGER;
  DN   Normalized DistinguishedName;
 }
 }
```

Each DN has its own return code. If the return code is not SUCCESS, a DN of zero length is returned for every DN passed in the original request. The order of DN values in the response matches the order of DN values that are passed in the request. The LDAP return code and their corresponding error condition for the extended operation is as follows:

- `Success`: The DN was normalized successfully.
- `UndefinedAttributeType`: An attribute in the DN is undefined.
- `InvalidDNSyntax`: The DN syntax is invalid.

Behavior

The extended operation normalizes a DN, or list of DNs. The normalization is based on the schema. See "`slapi_dn_normalize_v3`" in the *IBM Security Directory Server Server Plug-ins Reference.*

All users can run this extended operation.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_OTHER`
- `LDAP_OPERATIONS_ERROR`

This extended operation is supported by the Administration Server.

Scope The extended operation affects only the current operation.

Dynamic server trace extended operation

The dynamic server trace extended operation explains its use with the server and provides the results.

Description

Use this extended operation to do the following actions:

- Start or stop server-tracing dynamically
- Set the level of debug data collected
- Name the debug output file

This extended operation depends on the LDAP Trace Facility to be initialized with either the **`ldtrc`** command or the successful completion of the LDAP trace facility extended operation request on IBM Security Directory Server. See "LDAP trace facility extended operation" on page 224.

Note:

1. This extended operation is always enabled.
2. Global administrative group members have authority to run the dynamic server trace extended operation when it is directed to the directory server. However, global administrative group members do not have this authority when they request the extended operation against the Administration Server.

Request

OID 1.3.18.0.2.12.40

Syntax

The value consists of two integer values and an optional string. The first integer turns on tracing (`1`) or off (`0`). The second integer sets the debug level (`0` to `65535`) that controls the debug data that is directed to standard error (`stderr`) or a file. If the integers are missing, the request fails. If the value is `-1`, no change is made. The string value provides the file name and is optional. If no name is provided, the name is unchanged. If no name is ever provided, the debug output goes to `stderr`.

Response

OID 1.3.18.0.2.12.42

Syntax

The response is a string:

```
Trace settingsactual: enable=%d%d trcEvents=%ld%ld
 level=0x%x0x%x log=[%s]%s
```

where values in the brackets show that the state after the extended operation is attempted. If the tracing is on, `enable` is `1`. The `trcEvents` is `0` if the LDAP Trace Facility is not enabled. Non-zero values indicate that the server was successful in attaching to the LDAP Trace Facility shared memory buffer. The debug level is shown in hex. The log values is the name of the file that is used to collect the debug output. It might show `stderr` if the output is going to the console.

Behavior

This extended operation changes the global variables that are used to control debugging and tracing in the server. If trace is enabled but the debug level is `0`, trace data (function entry and exit points, and other data) is captured in shared memory and nothing is written to the debug file or `stderr`. If the debug level is between `0` to `65535`, different levels of debug data are output to the debug file or `stderr`. If the LDAP Trace facility is not initialized, no trace output is captured and no debug output is written.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_PROTOCOL_ERROR`

This extended operation is not supported by the Administration Server.

Scope Only the current server session is affected by this operation.

Auditing

The additional information in the audit log is:

```
Trace=%d [1=on|0=off] debug=0x%x log=[%s]
```

Dynamic update requests extended operation

The dynamic update request extended operation explains its use with the server and provides the results.

Description

The Dynamic update extended operation requests that the server reread its configuration.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.28

Syntax

```
RequestValue ::= SEQUENCE {
 action INTEGER {rereadFile(0),
         rereadAttribute(1),
         rereadEntry(2),
         rereadSubtree(3)};
 entry   [0] DistinguishedName OPTIONAL;
 attribute  [1] DirectoryString OPTIONAL;
}
```

Response

OID 1.3.18.0.2.12.29

Syntax

There is no response value.

Behavior

This extended operation forces the server to reread the configuration file. The request can be to reread the entire file, a subtree, an entry, or a specific attribute. When the server receives the request, the server reads the configuration file again. It updates all the internal server settings to use the new settings from the configuration file. Only the dynamic attributes are read again.

Only Primary Directory Administrator and Local Administration Group members with `DirDataAdmin` role are enabled to call this extended operation. Local Administration Group members cannot update attributes of other Local Administration Group members.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_UNDEFINED_TYPE`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_INVALID_SYNTAX`
- `LDAP_INVALID_DN_SYNTAX`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_OBJECT_CLASS_VIOLATION`
- `LDAP_OTHER`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_NO_SUCH_ATTRIBUTE`
- `LDAP_NO_SUCH_OBJECT`

- `LDAP_NO_MEMORY`

This extended operation is supported by the Administration Server.

Scope This extended operation causes the server to reread its configuration, which can affect subsequent operations.

Auditing
The Scope is provided along with the entry `dn`, or attribute when necessary.

`Scope:` *`Scope Value`*

where *Scope Value* can be one of the following values:

- Entire - entire configuration file
- Single - for a single attribute
- Entry - for an entry
- Subtree - for a subtree

`DN:` *`Entry DN`* – This DN is required for Single, Entry, and Subtree.

`Attribute:` *`Attribute`* – This attribute is required for Single only.

Effective password policy extended operation

The effective password policy extended operation explains its use with the server and provides the results.

Description
For a user in a DIT, there are three different password policies that can be used to control the user's login and password modifications. An administrator can use this extended operation to obtain users' effective password policy and manage users and their passwords. In addition, administrators can also use this extended operation to query the effective password policy of a group. By specifying a group DN in the operation, administrators can obtain a combination of the group password policy attributes and the global password policy attributes. The combination is with the group policy attribute values overriding the global policy values.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.75

Syntax

```
RequestValue ::= SEQUENCE {
       dn    LDAPDN
}
```

Response

OID 1.3.18.0.2.12.77

Syntax

```
ResponseValue ::= SEQUENCE {
     attributes     SEQUENCE OF SEQUENCE {
                   attributeType     AttributeDescription,
                   values            SET OF AttributeValue
     }
     objectNames    [0] SEQUENCE {
                   objectName        LDAPDN
     } OPTIONAL
}
```

where,

- `attributes`: Represents password policy attribute types and values that are contained in the user or group effective password policy.
- `objectNames`: Represents the DNs of all the password policy entries from which the effective password policy is derived. The **objectNames** field is not returned if the extended operation is requested by a non-administrative user.

Behavior

The information that is related to the effective password policy for a user or group is calculated at run time. It is not stored in the server, such as in the DIT. An administrator or a user can use this information, which is calculated on the three types of password policies global, group, and individual, to manage passwords.

This extended operation can be used by primary directory administrators, local administration group members with password or directory administrative role, and global administration group members. In additions, users are allowed to use this extended operation to their own effective password policy, provided their user account are not locked out. If the extended operation is called by a non-authorized user, a return code `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_INSUFFICIENT ACCESS` returned if a non-authorized user tries to run this extended operation
- `LDAP_NO_SUCH_OBJECT` returned if the DN specified is not in the directory
- `LDAP_NO_MEMORY` returned if there was insufficient memory to run the operation
- `LDAP_OPERATIONS_ERROR` returned if invalid data was given on the call to the password policy routines
- `LDAP_INVALID_DN_SYNTAX` returned if the DN specified is not a valid DN
- `LDAP_PROTOCOL_ERROR` returned if the encoding of the BER was invalid
- `LDAP_SUCCESS` returned if the operation completed successfully

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

The additional information in the audit log is:

`DN:` *`entry dn`*

End transaction extended operation

The end transaction extended operation explains its use with the server and provides the results.

Description

The End transaction extended operation sends requests to the server to do the following actions:

- Commit all the operations that are run inside the transaction
- Change the state of the connection so it is no longer in the transactional state

Note: This extended operation is enabled by default, but can be disabled by changing the value of the **`ibm-slapdTransactionEnable`** attribute in the configuration file.

The **`ibm-slapdTransactionEnable`** attribute is in the `cn=Transaction, cn=Configuration` entry in the configuration file. If the value of this attribute is set to `FALSE`, transactions are disabled. If the value is set to `TRUE`, transactions are enabled. Transactions can also be enabled or disabled by using the web administration tool.

Request

OID 1.3.18.0.2.12.6

Syntax

A string that consists of commit or rollback value followed by the transaction ID value from the Begin transaction response. The commit or rollback has the following values:

commit = 0

rollback = 1

Response

OID 1.3.18.0.2.12.6

Syntax

The response value is a string that contains the transaction ID. The transaction ID is not BER encoded.

Behavior

The extended operation commits the transaction and removes the connection from the transaction state.

All users can run this extended operation.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_TIMELIMIT_EXCEEDED`
- `LDAP_SIZELIMIT_EXCEEDED`

This extended operation is not supported by the Administration Server.

Scope This extended operation changes the state of the connection for future operations. The extended operation takes the connection out of the transactional state.

An Example

An example that illustrates the difference in the transaction ID value in a Begin transaction extended operation and an End transaction extended operation is exemplified.

If in a Begin transaction extended operation, the response value that is returned is the following 24 byte:

```
0x31 0x31 0x33 0x37 0x33 0x35 0x34 0x33##linebreak##0x33 0x37 0x31 0x32
0x37 0x2E 0x30 0x2E##linebreak##0x30 0x2E 0x31 0x34 0x38 0x39 0x30 0x38
```

In the End transaction extended operation, the request value for a commit (commit = 0) can be the following 25 byte:

```
0x00 0x31 0x31 0x33 0x37 0x33 0x35 0x34##linebreak##0x33 0x33 0x37 0x31
0x32 0x37 0x2E 0x30##linebreak##0x2E 0x30 0x2E 0x31 0x34 0x38 0x39
0x30##linebreak##0x38
```

In the End transaction extended operation, the request value for a rollback (rollback = 1) can be the following 25 byte:

```
0x01 0x31 0x31 0x33 0x37 0x33 0x35 0x34##linebreak##0x33 0x33 0x37 0x31
0x32 0x37 0x2E 0x30##linebreak##0x2E 0x30 0x2E 0x31 0x34 0x38 0x39
0x30##linebreak##0x38
```

Event notification register request extended operation

The event notification register request extended operation explains its use with the server and provides the results.

Description

The operation allows a client to request that the server notifies the client when a portion of the tree changes.

Note: Event notification can be turned off by setting the attribute **ibm-slapdEnableEventNotification** in the entry cn=Event Notification, cn=Configuration to FALSE.

Request

OID 1.3.18.0.2.12.1

Syntax

```
changeType ::= ENUMERATED {
   changeAdd      (1),
   changeDelete   (2),
   changeModify   (4),
   changeModDN    (8) }
 requestValue = SEQUENCE {
  eventID  ENUMERATED {
   LDAP_CHANGE (0)},
  baseObject LDAPDN,
              scope           ENUMERATED {
                      baseObject              (0),
                      singleLevel             (1),
                      wholeSubtree            (2) },
  type INTEGER OPTIONAL }
```

Response

OID 1.3.18.0.2.12.1

Syntax

```
response ::= OCTET STRING
```

Behavior

If successful, the server sends an unsolicited notification to the client when a modification happens that the client is interested in.

All users other than anonymous can run this extended operation.

This extended operation has the following possible return codes:

- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_SUCH_OBJECT
- LDAP_UNDEFINED_TYPE

This extended operation is not supported by the Administration Server.

Scope If successful, the client might receive unsolicited notifications from the server.

Auditing

```
eventID: LDAP_change
base: baseDn
scope: baseObject, singleLevel, or wholeSubtree
```

Event notification unregister request extended operation

The event notification unregister request extended operation explains its use with the server and provides the results.

Description

The operation allows a client to request that the server must stop notifying the client when a portion of the tree changes.

Note: Event notification can be turned off by setting the attribute **`ibm-slapdEnableEventNotification`** in the entry `cn=Event Notification, cn=Configuration` to `FALSE`.

Request

OID 1.3.18.0.2.12.3

Syntax

```
requestValue ::= OCTET STRING
```

Response

OID 1.3.18.0.2.12.4

Syntax

If the registration is successfully removed, the **LDAPResult** field contains `LDAP_SUCCESS` and the response field contains the registration ID that was removed.

Behavior

If successful, the server stops sending unsolicited notifications to the client when a modification happens that the client was interested in.

All users other than anonymous can run this extended operation.

This extended operation has the following possible return codes:

- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_UNDEFINED_TYPE`

This extended operation is not supported by the Administration Server.

Scope If successful, the client stops receiving unsolicited notifications from the server.

Auditing

```
ID: hostname.uuid
```

Group evaluation extended operation

The group evaluation extended operation explains its use with the server and provides the results.

Description

The Group evaluation extended operation requests that the server return the set of groups to which the requested user belongs.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.50

Syntax

```
GroupEvaluationRequestValue:: = SEQUENCE  {
    dn  LDAPDN,
                attributes AttributeList  OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.52

Syntax

```
Group ::= SEQUENCE { groupName LDAPString }
GroupEvaluationResponseValue :: = SEQUENCE{
      normalized  INTEGER{unnormzlied(0), normalized(1)};
      Sequence of Group }
```

Behavior

This extended operation determines to which groups the requested user belongs.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role
- Global Administrators

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_INVALID_DN_SYNTAX`
- `LDAP_NO_RESULTS_RETURNED`
- `LDAP_PROTOCAL_ERROR`
- `LDAP_NO_SUCH_ATTRIBUTE`

This extended operation is not supported by the Administration Server.

Scope The extended operation affects only the current operation.

Auditing

The group evaluation extended operation sets the audit string to

`DN:` *the DN sent in the group evaluation extended operation* `\n`

If **`ibm-auditAttributesOnGroupEvalOp`** is `TRUE`, the audit string contains a list of attribute value pairs that are separated by a new line. If the **`ibm-auditAttributesOnGroupEvalOp`** is `FALSE`, the string contains:

```
sentAttrs: true|false
```

The value is `FALSE` if no attributes were sent on the request.

Kill connection extended operation

The kill connection extended operation explains its use with the server and provides the results.

Description

The Kill connection extended operation requests that the server to stop the specified connections. Connections can be stopped based on the following parameters:

- Connection IP
- Connection DN
- Combination of IP and DN
- All connections

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.35

Syntax

```
ReqType ::= ENUMERATED {
DN (1),
IP (2)
}
RequestValue ::= SEQUENCE {
SET  {type  ReqType
              value  Directory String} OPTIONAL
SET  {type  ReqType
              value  Directory String} OPTIONAL
}
```

For a DN-specific or IP-specific request, only one set of type and value is needed. For a combination DN or IP request, both sets of type and value are needed. If there is no value that is specified, all connections are stopped.

Response

OID 1.3.18.0.2.12.36

Syntax

```
ResponseValue ::= { int numberKilled
                    int numberPending }
```

Each DN has its own return code. If the return code is not SUCCESS, a DN of zero length is returned for every DN passed in on the original request. The order of DN values in the response matches the order of DN values that are passed in the request.

Behavior

This extended operation stops the requested connections.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role
- Global Administrators

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_INVALID_DN_SYNTAX`
- `LDAP_OTHER`
- `LDAP_PROTOCAL_ERROR`

This extended operation is not supported by the Administration Server.

Scope The extended operation affects only the current operation.

Auditing
The DN or IP or both are provided:

```
DN: DN
IP: IP
```

If the DN or the IP is not present, then the request was to stop all connections.

LDAP trace facility extended operation

The LDAP trace facility extended operation explains its use with the server and provides the results.

Description
Use this extended operation to control LDAP trace facility remotely by using the Administration Server.

Note: This extended operation is always enabled on the Administration Server. It is not supported on the Directory Server.

Request

OID 1.3.18.0.2.12.41

Syntax
The value consists of one integer value and a string. The first integer has the following values:

- 1 enables the LDAP trace facility
- 2 disables the LDAP trace facility
- 3 enables changing masks or other parameters
- 4 clears data that is already collected in the shared memory buffer
- 5 shows information about the current state
- 6 creates a file from the data that is already captured in shared memory

The optional string contains more parameters that are understood by the **`ldtrc`** command, such as the size of the buffer (1) or the name of the output file for memory dump (6).

Response

OID 1.3.18.0.2.12.43

Syntax
The response is a string that contains the output from the **`ldtrc`** command that is submitted remotely.

Behavior

The extended operation submits an **ldtrc** command on the host computer and captures its output to return to the client.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with DirDataAdmin role

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_PROTOCOL_ERROR

This extended operation is supported by the Administration Server only.

Scope The extended operation runs until the computer is rebooted, the root manually issues IPC commands, the **ldtrc** command is issued on the computer, or another request is made.

Auditing

The additional information in the audit log is:

OPTIONS: *request valueoptional string*

where *request value* is the request value (1-6) and *optional string* is any additional parameters for **ldtrc**.

Locate entry extended operation

The locate entry extended operation explains its use with the server and provides the results.

Description

This extended operation is used to extract the back-end server details of a set of entry DNs and provide the details to the client.

Request

OID 1.3.18.0.2.12.71

Syntax

```
RequestValue ::= SEQUENCE {
  DN DistinguishedName;
            //a normalized DN is passed to a proxy server
}
```

Response

OID 1.3.18.0.2.12.72

Syntax

```
ResultValue ::= SEQUENCE {
  partitionInformationObject PIO; //depends on the access rights
}
```

where, the **partitionInformationObject** constitutes:

- split name
- partition base DN
- partition index

- server group
- list of the server URLs

Behavior

In a distributed directory setup, data are distributed across a set of back-end servers. Also, the back-end servers are made clear to the users, by placing a proxy server in front of this set back-end server. There are situations, where administrators might want to locate the back-end servers on which a set of entries exist. This extended operation can be used to extract the back-end server details of a set of entry DNs and provide the details to the client.

This extended operation for locating entries on the backend-servers can be only run by Primary Directory Administrator, Local Administration Group members with `DirDataAdmin` role, and Global Administration Group members. If a non-authorized user attempts to run this extended operation, `LDAP_INSUFFICIENT_ACCESS` is returned.

Note: There is no mechanism in place to restrict the administrators from locating the entries.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_INVALID_DN_SYNTAX`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_SERVER_DOWN`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_ENCODING_ERROR`
- `LDAP_DECODING_ERROR`

This extended operation is not supported by the Administration Server.

Scope This extended operation does not affect the subsequent operations on the connection.

LogMgmtControl extended operation

The `LogMgmtControl` extended operation explains its use with the server and provides the results.

Description

The `LogMgmtControl` extended operation is used to start, stop, and query the status of the log management for a directory server instance and proxy server instance that are running on a system.

Request

OID 1.3.18.0.2.12.70

Syntax

```
requestValue ::= SEQUENCE {
        action              ActionValue,
        commandLineOptions  LDAPString OPTIONAL
}
 ActionValue ::= ENUMERATED {
   start    (1),
   stop     (2),
   status   (3)
 }
```

Response

Syntax

```
ResponseValue ::= SEQUENCE {
    resultCode       ENUMERATED {
         success                  (0),
         insufficientAccessRights   (1),
         operationsError          (2),
         logmgmtRunningStatus       (3),
         logmgmtStoppedStatus       (4)
   }
}
```

The possible return codes for the LDAP result value and the enabling conditions are as follows:

Table 12. Possible return codes

LDAP Result Value	Conditions
Success (0)	Issued command to `idslogmgmt` successfully.
Insufficient Access Rights (1)	User is not the server administrator or an administrative group member.
Operations Error (2)	Bad action value or any other error.
Log Management Running Status (3)	The log management for the directory server instance is running.
Log Management Stopped Status (4)	The log management for the directory server instance is stopped.

Behavior

The `LogMgmtControl` extended operation can be used to start or stop the log management for a directory server instance and proxy server instance. This extended operation also provides the status of the log management that indicates whether it is running or not.

The following have the authority to call this extended operation:

- Primary Directory Administrator
- Local Administration Group members with `AuditAdmin` and `ServerConfigGroupMember` roles

This extended operation is supported by Administration Server and has the same behavior as in a directory server.

Scope Only the current server session is affected by this operation.

Online backup extended operation

The online backup extended operation explains its use with the server and provides the results.

Description

This extended operation runs online backup of the directory server instance DB2 database.

Request

OID 1.3.18.0.2.12.74

Syntax

```
RequestValue ::= SEQUENCE {
      directoryPath directoryString;
}
```

Response

OID 1.3.18.0.2.12.74

Syntax

```
ResponseValue ::= SEQUENCE {
      resultCode    INTEGER (0..MAX)
}
```

Behavior

A client sends the online backup request to the server for running an online backup of the directory server instance DB2 database.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role

Note: If the extended operation is called by a user who does not have the required access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS` - If the backup was successfully run.
- `LDAP_PROTOCOL_ERROR` - If there is an error in the format of the request.
- `LDAP_INSUFFICIENT_ACCESS` - If the request is from users who do not have the required access.
- `LDAP_OPERATIONS_ERROR` - Internal Server error, database is not configured for online backup.
- `LDAP_NO_SUCH_OBJECT` - The specified directory path does not exist.

This extended operation is not supported by the Administration Server.

Scope The extended operation affects only the current operation.

Auditing

The following information is audited for this extended operation:

`Online backup requested:` *`directoryPath`*

Password policy bind initialize and verify extended operation

The password policy bind-initialize and verify extended operation explains its use with the server and provides the results.

Description

This extended operation runs password policy bind initialization and verification for a specified user. This extended operation checks to see whether an account is locked. The extended operation provides a mechanism for the proxy server to support bind plug-ins.

Request

OID 1.3.18.0.2.12.79

Syntax

```
requestValue ::= SEQUENCE {targetDN DirectoryString}
```

Response

OID 1.3.18.0.2.12.79

Syntax

```
responseValue ::= SEQUENCE {ReturnCode Integer}
```

Behavior

This extended operation runs `prebind` processing that is related to password policy, that is, bind initialization and verification for a specified user. This extended operation also checks whether an account is locked. The extended operation provides a mechanism for the proxy server to support bind plug-ins. This extended operation can be enabled or disabled by setting the **`ibm-slapdEnableRemotePWPExOps`** attribute in the configuration file to `TRUE` or `FALSE` in the following entry:

```
"cn=Directory, cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration"
```

The following persons are enabled to call this extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role
- Global administration group members

Note: If this extended operation is called by a user who does not have enough access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS` - The operation is completed successfully, caller must check the return code in the result value.
- `LDAP_OPERATIONS_ERROR` - The operation did not complete successfully because of an internal server error. There is not any result value.
- `LDAP_INSUFFICIENT_ACCESS` - The operation did not complete because the requestor does not have permission to run the operation. There is not any result value.
- `LDAP_UNWILLING_TO_PERFORM` - The user account is locked.
- `LDAP_INVALID_CREDENTIALS` - Invalid DN or password.
- `LDAP_NO_MEMORY`

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

The additional information in the audit log for this extended operation is listed. The target DN is audited in the following format:

`targetDN:` *`DN value`*

Password policy finalize and verify bind extended operation

The password policy-finalize and verify bind extended operation explains its use with the server and provides the results.

Description

This extended operation runs password policy post-bind processing for a specified user. This extended operation provides a mechanism for the proxy server to support bind plug-ins. Post bind processing includes checking for expired passwords, grace logins, and updating failed and successful bind counters.

Request

OID 1.3.18.0.2.12.80

Syntax

```
requestValue ::= SEQUENCE {
            targetDN      DirectoryString,
            bindResult    Integer
}
```

Response

OID 1.3.18.0.2.12.80

Syntax

```
ResponseValue ::= SEQUENCE {
          PasswordEvaluationReturnCode    Integer
}
```

Behavior

The password policy-finalize and verify bind extended operation runs all the post-bind processing that is related to password policy. It checks for expired accounts and grace login period. In addition, failed and successful bind counts are updated for the target entry.

This extended operation can be enabled or disabled by setting the **ibm-slapdEnableRemotePWPExOps** attribute in the configuration file to TRUE or FALSE in the following entry:

```
"cn=Directory, cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration"
```

The following persons are enabled to call this extended operation:

- Primary Directory Administrator
- Local administration group members with DirDataAdmin role
- Global administration group members

Note: If this extended operation is called by a user who does not have enough access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS - The operation is completed successfully, caller must check the return code in the result value.
- LDAP_OPERATIONS_ERROR - The operation did not complete successfully because of an internal server error. There is not any result value.
- LDAP_INSUFFICIENT_ACCESS - The operation did not complete because the requestor does not have permission to run this operation. There is not any result value.
- LDAP_UNWILLING_TO_PERFORM - The user account is locked.
- LDAP_INVALID_CREDENTIALS - Invalid DN or password.
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

The additional information in the audit log for this extended operation is listed. The target DN and bind result is audited in the following format:

```
targetDN: targer DN
bindResult: rc
```

Prepare transaction extended operation

The prepare transaction extended operation explains its use with the server and provides the results.

Description

The prepare transaction extended operation can be sent by any client. Using this extended operation, the client requests the server to start processing the operations that are sent in a transaction. This extended operation must be called after a start transaction is issued and all the operations within a transaction are sent. On getting a request, the server starts processing each operation without committing the changes. This extended operation is enabled only when transactions are enabled.

Request

OID 1.3.18.0.2.12.64

Syntax

```
requestValue ::= { transactionId  String; }
```

Response

OID 1.3.18.0.2.12.64

Syntax

This extended operation returns the return code for the operation.

Behavior

When the server receives the extended operation, the server checks whether the connection is in the transactional state and no commit or prepare request are sent. If these checks pass, the server starts processing each operation in the transaction without a commit. This extended operation is not supported by the Administration Server.

Auditing

No additional auditing information is provided for this operation.

Note: There is no requirement to audit the transaction ID because this value is already audited when it is sent by using the transaction control.

Proxy back-end server resume role extended operation

The proxy back-end server resume role extended operation explains its use with the server and provides the results.

Description

This extended operation enables a proxy server to resume the configured role of a back-end server in a distributed directory environment.

Request

OID 1.3.18.0.2.12.65

Syntax

```
requestValue ::= SEQUENCE {
  action     ENUMERATED {
                          All (0),
                         Partition (1),
                         Server (2),
                         ServerInAPartition (3)
         };
  PartitionName       DirectoryString;
  ServerURL           DirectoryString;
}
```

Response

OID 1.3.18.0.2.12.65

Syntax

```
responseValue ::= SEQUENCE {
  numObjectsImpacted      INTEGER
}
```

Behavior

This extended operation tells a proxy server to bring a back-end server back to its configured role. The proxy server resumes only a back-end server role if the back-end server is online and accepting connections from the proxy server. The extended operation uses the 5 second reconnect interval or the health check thread to connect with the back-end server.

The following are authorized to call this extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` role
- Global Administration Group members

Note: If this extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS` - Server matched the request, and no internal errors were encountered.
- `LDAP_PROTOCOL_ERROR` - If there is an error in the format of the request.
- `LDAP_INSUFFICIENT_ACCESS` - If the request is from a user who does not have the required access.
- `LDAP_OPERATIONS_ERROR`
- `LDAP_NO_SUCH_OBJECT` - If the requested target does not exist.
- `LDAP_INVALID_SYNTAX` - If the format of the URL or partition name is invalid.

This extended operation is not supported by the Administration Server.

Scope This extended operation affects requests are routed through the proxy server.

Auditing

The additional information added to audit log by the proxy back-end server resume role extended operation are:

`RequestType:` *`Type`*

where *Type* is one of the following types:

- All
- Partition
- Server
- ServerInAPartition

`Partition:` *`PartitionName`*

`Server:` *`ServerURL`*

Note: If *PartitionName* or *ServerURL* is not specified in the request, `None` is audited.

Quiesce or unquiesce replication context extended operation

The quiesce or unquiesce replication context extended operation explains its use with the server and provides the results.

Description

This extended operation is used for the following changes:

- Disable non-replication topology-related changes in the replication context.
- Enable non-replication topology-related changes.

Request

OID 1.3.18.0.2.12.19

Syntax

```
requestValue ::= SEQUENCE {
quiesce BOOLEAN,
subtreeDn DistinguishedName
}
```

Response

OID 1.3.18.0.2.12.19

Syntax

```
ResponseValue ::= SEQUENCE {
#fields of interest from LDAPResult:
resultCode INTEGER (0..MAX),
errorMessage LDAPString,
}
```

The following return codes are possible:

- `LDAP_SUCCESS`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_NO_MEMORY`
- `LDAP_REPL_QUIESCE_BAD_STATE`

Behavior

This extended operation is used for the following changes:

- Disable non-replication topology-related changes in the replication context.
- Enable non-replication topology-related changes.

If the quiesce Boolean is `TRUE`, then only replication topology-related changes are enabled.

The following persons are enabled to call the extended operation:

- Primary Directory Administrator
- Local Administration Group members with `DirDataAdmin` and `ReplicationAdmin` roles
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_DECODING_ERROR`
- `LDAP_NO_MEMORY`
- `LDAP_UNDEFINED_TYPE`
- `LDAP_INVALID_DN_SYNTAX`

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

```
Action: [ Quiesce | Unquiesce ]
Context DN: dn
```

Replication error log extended operation

The replication error log extended operation explains its use with the server and provides the results.

Description

Use this extended operation to monitor replication errors and correct any problems that occur as data fails to be replicated.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.56

Syntax

The value consists of an integer which indicates the type of request and two strings in BER format. The first string identifies which failure or failures are to be deleted, attempted again or displayed. The value is either `0` for all, or the ID of the failed change. The second string provides the DN for the replication agreement.

Response

OID 1.3.18.0.2.12.57

Syntax

The response is a string that indicates any problem that occurred, or if successful, how many failed changes were deleted or present to the consumer.

Behavior

The extended operation acts on the table that maintains the updates that failed on any of the current server consumer servers. The data for any single failure can be displayed. Any or all failed changes can be deleted or attempted again. Deleted changes are removed from the table. Changes that attempted again are sent individually to the consumer. If the update succeeds, the failure is removed from the table. If the update fails again, it is added back as a new failure with the following results to reflect this update:

- number of attempts

- last time that attempted
- result code that is updated

The original failure is removed. The worker thread that handles the extended operation connects to the consumer and sends these changes. Replica threads can send updates to the consumer at the same time.

The following persons are enabled to call the extended operation:
- Primary Directory Administrator
- Local Administration Group member
- Users with write access to the replica group

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:
- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_DECODING_ERROR
- LDAP_PROTOCOL_ERROR
- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_SUCH_OBJECT

This extended operation is not supported by the Administration Server.

Scope If the errors are deleted or successfully attempted again, they are removed from the table permanently.

Auditing

The additional information in the audit log is consists of three lines:

```
Replication Error Log Management Option: [ SHOW | RETRY | DELETE | UNKNOWN ]
Replication Error ID: numeric value
Replication Agreement DN: DN or empty string.
```

Replication topology extended operation

The replication topology extended operation explains its use with the server and provides the results.

Description

This extended operation propagates replication topology-related entries from a supplier to the consumers in the network. This extended operation is useful to synchronize replication topology data for every server in the network before replication of directory entries can begin.

Request

OID 1.3.18.0.2.12.54

Syntax

```
RequestValue ::= SEQUENCE {
 replicationContextDn   DistingushedName,
 timeout    INTERGER,
 replicationAgreementDn DistingushedName OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.55

Syntax

```
ResponseValue ::= SEQUENCE {
 resultCode   INTEGER(0..MAX),
 errorMessage   LDAPString,
 #operation specific failure information:
 supplier  LDAPString,
 consumer  LDAPString,
}
```

Behavior

A supplier gathers its replication topology-related entries under a replication context and propagates them to the consumer servers. The supplier can add the entries to the consumer or modify the existing entries on the consumer or delete the extra entries from the consumer. As a result of the extended operation, the replication topology-related entries under the specified context on both the supplier and the consumers are in sync.

The operation is enabled when the client is authenticated with update authority to all agreements in the specified subtree. Or, it is authenticated as a master server for the specified subtree. Primary Directory Administrator and Local Administration Group members with `DirDataAdmin` and `ReplicationAdmin` roles are authorized to call this extended operation.

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_PROTOCOL_ERROR`

This extended operation is not supported by the Administration Server.

Scope The extended operation does not affect subsequent operation on the connection.

Auditing

Context DN, Replication Agreement DN, and Timeout are audited.

ServerBackupRestore extended operation

The `ServerBackupRestore` extended operation explains its use with the server and provides the results.

Description

The `ServerBackupRestore` extended operation issues request to the administration server to back up a directory server data and configuration files or restore a directory server data and configuration files from an existing backup.

Request

OID 1.3.18.0.2.12.81

Syntax

```
requestValue ::= SEQUENCE {
        action ActionValue
}
```

```
ActionValue ::= ENUMERATED {
       backup    (1),
       restore   (2)
}
```

Response

OID 1.3.18.0.2.12.81

Syntax

```
responseValue ::= SEQUENCE {
       result OperationResult
}

OperationResult ::= ENUMERATED {
       No_Operation_Attempted (1),
       Backup_Submitted (2),
       Failed_Backup_Requires_Server_Stop (3),
       Failed_Backup_In_Progress (4),
       Unknown_Backup_Result (5),
       Restore_Submitted (6),
       Failed_Restore_Requires_Server_Stop (7),
       Failed_Restore_In_Progress (8),
       Failed_Restore_No_Backup_Exists (9),
       Unknown_Restore_Result (10),
       Failed_Backup_Requires_Onetime_Server_Stop (11)
}
```

Behavior

This extended operation requests the administration server to do the following actions:

- Back up a directory server data and configuration files
- Restore a directory server data and configuration files from an existing backup that depends on the action value

This extended operation can be disabled by setting the **ibm-slapdBackupEnabled** attribute in the server configuration file to FALSE.

This extended operation has the following possible return codes:

- LDAP_SUCCESS - If the request is submitted successfully.
- LDAP_INSUFFICIENT_ACCESS - If the bind DN does not have the required permission to send request.
- LDAP_PROTOCOL_ERROR - If not an administration server, a directory server with RDBM, not configured for backup and restore, or backup and restore not enabled.
- LDAP_OPERATIONS_ERROR - If the action value is unsupported or missing.

The backup and restore operations can be only run by the following users:

- Primary directory administrator
- Local administration group member that has DirDataAdmin, ServerStartStopAdmin, ServerConfigGroupMember, and SchemaAdmin roles

This extended operation is only supported by the Administration Server.

Scope At a time only one operation can be run. If a backup or restore operation is running, other backup or restore requests results in error unless the earlier operation is completed. If a bulkload operation is running, then backup or restore operation does not proceed.

Auditing

```
Action: [backup|restore]
Response: [ 1 - No Operation Attempted |
            2 - Backup_Submitted |
            3 - Failed_Backup_Requires_Server_Stop |
            4 - Failed_Backup_In_Progress |
            5 - Unknown_Backup_Result |
            6 - Restore_Submitted |
            7 - Failed_Restore_Requires_Server_Stop |
            8 - Failed_Restore_In_Progress |
            9 - Failed_Restore_No_Backup_Exists |
           10 - Unknown_Restore_Result |
           11 - Failed_Backup_Requires_Onetime_Server_Stop
         ]
```

Start, stop server extended operations

The start and stop server extended operation explains its use with the server and provides the results.

Description

The Start, stop server extended operation, when sent to the Administration Server, requests that the Administration Server does the following actions:

- Start
- Stop
- Restart
- Give the status of the LDAP server
- Stop the Administration Server

The Start Stop Server Extended Operation, when sent to the LDAP Server, requests that the LDAP Server stop.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.26

Syntax

```
actionType ::= ENUMERATED {
   startServer  (0),
   stopServer  (1),
   restartServer  (2),
   serverStatus  (3),
   admStop                (4)}

 requestValue :: = SEQUENCE {
  action  actionType
  command options  string OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.27

Syntax

```
ResultValue :: SEQUENCE {
   Status  Integer
   ErrorString String
   }
```

Behavior

When sent to the Administration Server, the request does one of the following actions:

- Start
- Restart
- Stop
- Request the server status
- Stop the Administration Server

When sent to the LDAP Server, the server acknowledges only the request to stop the server. Any other request sent to the LDAP Server results in a return code of `LDAP_UNWILLING_TO_PERFORM`.

When the request is sent to the Administration Server, only a Primary Directory Administrator or Local Administration Group members with `ServerStartStopAdmin` role has the authority to make the request.

When the request is sent to the LDAP server, only Primary Directory Administrator, Local Administration Group members with `ServerStartStopAdmin` role, or a global administration group member has the authority to make the request.

This control has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_OTHER`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_PROTOCAL_ERROR`

This extended operation is supported by the Administration Server. This extended operation with the stop request is supported in the LDAP Server.

Scope The extended operation affects only the current operation, unless the request is to stop Administrator Server.

Auditing
In the LDAP server, the additional information contains:

```
Operation: Start | Stop | Restart | Admin Stop | Status
```

In the Administration Server, the additional information contains:

```
Operation: Start | Stop | Restart | Admin Stop | Status
```

On a start or restart operation the following line is audited:

```
Options: Additional Value
```

For example, a request to start the server with the `-a` option audits the following operation:

```
Operation: Start
Options: ---a
```

Start TLS extended operation

The start TLS extended operation explains its use with the server and provides the results.

Description
This extended operation requests that the server start by using encrypted communications over the connection.

Note: This extended operation is always enabled.

Request

OID 1.3.6.1.4.1.1466.20037

Syntax
There is no request value for the extended operation.

Response

OID 1.3.6.1.4.1.1466.20037

Syntax

- `LDAP_SUCCESS`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_PROTOCOL_ERROR`

Behavior
The extended operation is used to request that communication on the connection must be encrypted. The server expects a TLS handshake on the connection.

All Local Administration Group members irrespective of their roles and all users can run this extended operation.

Note: If the extended operation is called by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_PROTOCOL_ERROR`

This extended operation is supported by the Administration Server.

Scope When a TLS handshake is run, all communication on the connection is encrypted until a TLS closure alert is sent or the connection is closed.

Unique attributes extended operation

The unique attributes extended operation explains its use with the server and provides the results.

Description
The unique attributes extended operation provides a list of all non-unique (duplicate) values for a particular attribute.

Note: This extended operation can be disabled. Commenting out or removing the statement in the configuration file for the unique attribute extended operation plug-in disables this extended operation. For example, commenting out the statement:

```
ibm-slapdPlugin: extendedop /bin/libback-rdbm.dll initUniqueAttr
```

from the configuration file disables this extended operation on Windows systems.

Request

OID 1.3.18.0.2.12.44

Syntax

```
ExtendedRequest ::= SEQUENCE {
  requestName LDAPOID // OID for the IBM Unique Attributes
  requestValue LDAPOID // OID for an attribute requiring uniqueness
}
```

where *LDAPOID* is an OCTET STRING.

Response

OID 1.3.18.0.2.12.45

Syntax

```
ExtendedResponse ::= SEQUENCE {
 COMPONENTS OF LDAPResult,
 responseName   LDAPOID // OID for the IBM Unique Attributes
 Response     AttributeValueList // list of all
   conflicting attribute values
}
```

where *AttributeValueList* is a SEQUENCE OF AttributeValue and *LDAPOID* is an OCTET STRING.

Behavior

The extended operation lists all non-unique values for a particular attribute.

The following persons are enabled to call the extended operation:
- Primary Directory Administrator
- Local Administration Group members with DirDataAdmin role
- Global Administration Group members
- Master Server DN

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:
- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY
- LDAP_PARAM_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_OTHER

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

User type extended operation

The user type extended operation explains its use with the server and provides the results.

Description

This extended operation can be used by a bound user to determine the user type and roles the user has on a directory server instance. Without the extended operation, there is no programmatic way to determine the general capabilities of a user and where the user DN and password is stored.

It is possible for a user to belong to a user type and have different capabilities and store passwords under different types of entries or attributes.

Additionally, the extended operation distinguishes the root administrator from an administrative group member when a client must use the Administration Server to authenticate a user.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.37

Syntax
There is no request value for the extended operation.

Response

OID 1.3.18.0.2.12.38

Syntax

```
ResponseValue ::= SEQUENCE {
               STRING (UserType)
               INTEGER (Number of UserRoles)
               SEQUENCE OPTIONAL
                        {
                          STRING (UserRole)
                        }
}
```

Behavior
This extended operation can be used by a bound user to determine the user type and roles the user has on a directory server instance.

All users, including anonymous, are enabled to send the control.

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_NO_RESULTS_RETURNED`
- `LDAP_PROTOCAL_ERROR`

This extended operation is supported by the Administration Server.

Scope The extended operation affects only the current operation.

Log access extended operations

The log access extended operation explains its use with the server and provides the results.

Three types of extended operation requests support access to the log files. IBM Security Directory Server administration server supports the following log access extended operations:

- "Clear log extended operation" on page 243
- "Get lines extended operation" on page 245
- "Get number of lines extended operation" on page 246

The server provides access to the following log files:

- `ibmslapd.log`

- `db2cli.log`
- `db2clicmds.log`
- `audit.log`
- `bulkload.log`
- `ibmdiradm.log`
- `lostandfound.log`
- `idstools.log`
- `db2load.log`
- `tracemsg.log`
- `adminAudit.log` (this file is available only if the Administration Server audit log OID (1.3.18.0.2.32.11) is in the list of supported capabilities in the root DSE)
- `ibmslapd.trace.log` (this file is available only if the trace log OID (1.3.18.0.2.32.14) is in the list of supported capabilities in the root DSE)

Lines are numbered starting with line 0. A line is considered all characters up to and including a new line or 400 characters, whichever comes first.

To make the log access request, a client application can use the client APIs for extended operations. An LDAP v3 extended operation request has the form:

```
 ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
                requestName      [0] LDAPOID,
                requestValue     [1] OCTET STRING OPTIONAL }
```

All the extended requests use a LogType. LogType is defined as:

```
LogType ::= ENUMERATED {
 SlapdErrors       (1),
 CLIErrors        (2),
 AuditLog         (4),
 BulkloadLog      (8),
 AdminErrors      (16),
 AdminAudit       (32),
 DebugOutputFile  (64),
}

RequestValue ::= { log LogType; }
```

Clear log extended operation

The Clear log extended operation explains its use with the server and provides the results.

Description

The Clear log extended operation requests that the server clear the requested log. When the log is cleared, a line is written to the log file with the date and time that states when the log file was cleared.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.20

Syntax

```
RequestValue ::= { log LogType; }
```

Response

OID 1.3.18.0.2.12.21

Syntax
There is no response value.

Behavior
The extended operation clears the requested log file and writes a message in the log. It writes the date and time that states when the log was cleared.

Only the Primary Directory Administrator or Local Administration Group members with `AuditAdmin` and `ServerConfigGroupMember` roles are authorized to call this extended operation. Only the Primary Directory Administrator can clear the audit log. A Local Administration Group member does not have access to clear the audit log.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_NO_MEMORY`

This extended operation is not supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing
`Log:` *`Log name`*

Get file extended operation

The get file extended operation explains its use with the server and provides the results.

Description
This extended operation returns the contents of a file on the server.

Request

OID 1.3.18.0.2.12.73

Syntax

```
RequestValue ::= SEQUENCE {
        fileNumber INTEGER {Other(0),
                         V3.ibm.at(1), V3.ibm.oc(2),
                         V3.user.at(3), V3.user.oc(4),
                         V3.config.at(5), V3.config.at(6),
                         V3.system.at(7), V3.system.oc(8),
                         V3.modifiedschema(9), V3.ldapsyntaxes(10),
                         V3.matchingrules(11),
                         KeyRingFile(12), KeyDBFile(13)};
       fileName String;
}
```

Response

OID 1.3.18.0.2.12.73

Syntax

```
ResponseValue ::= SEQUENCE {
        length    INTEGER, // The length of the file.
        lines     OCTET STRING // The lines from the file.
}
```

Behavior

A client uses the get file request to retrieve the contents of schema-related files or SSL-related files from the server. If the connection between the client and server is not over SSL, the SSL-related files are not returned.

The Primary Directory Administrator is enabled to call the extended operation.

Note: If the extended operation is called by a user who does not have the required access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This extended operation has the following possible return codes:

- `LDAP_SUCCESS` - If the file was successfully read.
- `LDAP_PROTOCOL_ERROR` - If there is an error in the format of the request.
- `LDAP_INSUFFICIENT_ACCESS` – If the request is from users other than the administrators.
- `LDAP_OPERATIONS_ERROR` - Internal Server error.
- `LDAP_NO_SUCH_OBJECT` - The requested file does not exist.

This extended operation is not supported by the Administration Server.

Scope The extended operation affects only the current operation.

Auditing

The following information is audited for this extended operation:

```
File: [fileName | V3.ibm.at | V3.ibm.oc |
       V3.user.at | V3.user.oc |
       V3.config.at | V3.config.at |
       V3.system.at | V3.system.oc |
       V3.modifiedschema | V3.ldapsyntaxes |
       V3.matchingrules]
```

Get lines extended operation

The get lines extended operation explains its use with the server and provides the results.

Description

The Get lines extended operation requests that the server read the specified lines from the requested log and return them to the client.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.22

Syntax

```
RequestValue :== SEQUENCE
 {
  Log    LogType;
  firstLine   INTEGER;
  lastLine   INTEGER;
 }
```

Response

OID 1.3.18.0.2.12.23

Syntax

There is a response value only if the return code is `LDAP_SUCCESS`.

Behavior

This extended operation reads the requested set of lines from the requested file and returns the lines to the user.

Only the Primary Directory Administrator and Local Administration Group members with any roles other than `NoAdmin` role are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_NO_MEMORY`

This extended operation is supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

`Log:` *`Log name`*

Get number of lines extended operation

The Get number of lines extended operation explains its use with the server and provides the results.

Description

The Get number of lines extended operation requests that the server determine the number of lines in the requested log file.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.24

Syntax

```
LogType ::= ENUMERATED {SlapdErrors  (1),
   CLIErrors  (2),
   AuditLog  (4),
   BulkloadLog  (8),
   AdminErrors  (16),
                     AdminAudit             (32),
                     DebugOutputFile        (64),
                     LostAndFound           (128).
                     ConfigToolsLog         (256)}
 RequestValue ::= { log LogType; }
```

Response

OID 1.3.18.0.2.12.25

Syntax

```
ResponeValue:: = number of lines
```

Behavior

The extended requests that the server read the log file and determine the number of lines in the requested log file.

Primary Directory Administrator and Local Administration Group members with any roles other than `NoAdmin` role are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_NO_MEMORY`

This extended operation is supported by the Administration Server.

Scope This extended operation affects only the current operation.

Auditing

`Log:` *`Log name`*

See the section "Creating the administrative group" in the *Administering* section in the IBM Security Directory Server documentation to know more about the following information:
- Administrative roles
- Authorization that is required to issue various extended operations
- Permissions that are required to access various objects

OIDs for controls

The OIDs for controls provide support description for various servers.

The following table shows OIDs for controls. Click the short name or go the specified page number for more information about a control syntax and usage.

Table 13. OIDs for controls

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"AES bind control" on page 251 1.3.18.0.2.10.28	This control enables the directory server to send updates to the consumer server with passwords already encrypted using AES.	No	Yes	No	No

Table 13. OIDs for controls (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Audit control" on page 252 1.3.18.0.2.10.22	The control sends a sequence of `uniqueid` strings and a source `ip` string to the server. When the server receives the control, it audits the list of `uniqueids` and `sourceip` in the audit record of the operation.	Yes	Yes	Yes	Yes
"Do not replicate control" on page 253 1.3.18.0.2.10.23	This control can be specified on an update operation (add, delete, modify, `modDn`, `modRdn`).	No	Yes	No	No
"Entry change notification control" on page 254 2.16.840.1.113730.3.4.7	This control provides more information about the changes that caused a particular entry to be returned as the result of a persistent search.	No	Yes	No	No
"Group authorization control" on page 255 1.3.18.0.2.10.21	The control sends a list of groups that a user belongs to.	No	Yes	No	No
"LDAP delete operation timestamp control" on page 256 1.3.18.0.2.10.32	This control is used to send the modified timestamp values to a replica during a delete operation.	No	Yes	No	No
"Limit number of attribute values control" on page 257 1.3.18.0.2.10.30	This control limits the number of attribute values that are returned for an entry in a search operation.	No	Yes	Yes	Yes
"Manage DSAIT control" on page 258 2.16.840.1.113730.3.4.2	Causes entries with the `ref` attribute to be treated as normal entries, allowing clients to read and modify these entries. * In IBM Security Directory Server Proxy Server (without partitioned data), even if this control is not included in the request the proxy server always sends the `Manage DSAIT` control to the back-end server.	No	Yes	Yes (*)	No

Table 13. OIDs for controls (continued)

Short name with OID	**Description**	**Supported by the Administration Server**	**Supported by IBM Security Directory Server full server 6.3.1 with database**	**Supported by IBM Security Directory Server Proxy Server 6.3.1**	
				Without partitioned data	**With partitioned data**
"Modify groups only control" on page 259 1.3.18.0.2.10.25	Attached to a delete or modify DN request to cause the server to do only the group referential integrity processing. The processing is for the delete or rename request without doing the actual delete or rename of the entry itself. The entry that is named in the delete or modify DN request does not require to exist on the server.	No	Yes	No	No
"No replication conflict resolution control" on page 260 1.3.18.0.2.10.27	When present, a replica server accepts a replicated entry without trying to resolve any replication conflict for this entry.	No	Yes	No	No
"Omit group referential integrity control" on page 260 1.3.18.0.2.10.26	Omits the group referential integrity processing on a delete or `modrdn` request. When present on a delete or rename operation, the entry is deleted from or renamed in the directory. But the entry membership is not removed or renamed in the groups in which the entry is a member.	No	Yes	No	No
"Paged search results control" on page 261 1.2.840.113556.1.4.319	Allows management of the amount of data that is returned from a search request.	No	Yes	Yes	Yes
"Password policy request control" on page 262 1.3.6.1.4.1.42.2.27.8.5.1	Password policy request or response	Yes	Yes	Yes	Yes
"Persistent search control" on page 264 2.16.840.1.113730.3.4.3	This control provides clients a means to receive notification of changes in the LDAP server.	No	Yes	No	No

Table 13. OIDs for controls (continued)

Short name with OID	**Description**	**Supported by the Administration Server**	**Supported by IBM Security Directory Server full server 6.3.1 with database**	**Supported by IBM Security Directory Server Proxy Server 6.3.1**	
				Without partitioned data	**With partitioned data**
"Proxy authorization control" on page 265 2.16.840.1.113730.3.4.18	The Proxy Authorization Control enables a bound user to assert another user identity. The server uses this asserted identity in the evaluation of ACLs for the operation.	No	Yes	No	No
"Refresh entry control" on page 266 1.3.18.0.2.10.24	This control is returned when a target server detects a conflict during a replicated modify operation.	No	Yes	No	No
"Replication bind failure timestamp control" on page 266 1.3.18.0.2.10.34	The master server uses the replication bind failure timestamp control to propagate the bind failure timestamp value to a read-only replica server.	No	Yes	No	No
"Replication supplier bind control" on page 268 1.3.18.0.2.10.18	This control is added by the supplier, if the supplier is a gateway server.	No	Yes	No	No
"Replication update ID control" on page 269 1.3.18.0.2.10.29	This control was created for serviceability. If the supplier server is set to issue the control, each replicated update is accompanied by this control.	No	Yes	No	No
"Return deleted objects control" on page 269 1.3.18.0.2.10.33	This control when included in a null base search request, all entries in the database that includes those entries with attribute `isDeleted` set to `TRUE` are returned.	No	Yes	No	No
"Server administration control" on page 270 1.3.18.0.2.10.15	Allows an update operation by the administrator under conditions when the operation would normally be refused (server is quiesced, a read-only replica, and others). * In IBM Security Directory Server Proxy Server, this control is supported only for bind operations.	Yes	Yes	Yes (*)	Yes (*)

Table 13. OIDs for controls (continued)

Short name with OID	Description	Supported by the Administration Server	Supported by IBM Security Directory Server full server 6.3.1 with database	Supported by IBM Security Directory Server Proxy Server 6.3.1	
				Without partitioned data	With partitioned data
"Sorted search results control" on page 272 1.2.840.113556.1.4.473	Allows a client to receive search results that are sorted by a list of criteria, where each criterion represents a sort key.	No	Yes	No	No
"Subtree delete control" on page 273 1.2.840.113556.1.4.805	This control is attached to a Delete request to indicate that the specified entry and all descendant entries are to be deleted.	No	Yes	No	No
"Transaction control" on page 273 1.3.18.0.2.10.5	Marks the operation as part of a transactional context. * In IBM Security Directory Server Proxy Server, transactions are supported only when all updates target a single partition.	No	Yes	Yes (*)	Yes (*)
"Virtual list view control" on page 274 2.16.840.1.113730.3.4.9	This control extends the regular LDAP search operation and includes a server-side sorting control.	No	Yes	No	No

AES bind control

The AES bind control explains its use with the server and provides the results.

Description
This control enables the directory server to send updates to the consumer server with passwords already encrypted by using AES. If the consumer server does not support AES encryption of passwords, or the seed or salt values do not match, the directory server decrypts the `userpassword` and `secretkey` values in updates to be replicated.

Note: This control is always enabled.

OID 1.3.18.0.2.10.28

Syntax
This control has no value.

Behavior

The criticality must be set to `TRUE` to protect clients from submitting a request with an unauthorized identity.This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Bind

The following persons are enabled to send the control:

- Primary Directory Administrator
- Master Server DN
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
This control has the possible return code as `LDAP_INSUFFICIENT_ACCESS`.
This control is not supported by the Administration Server.

Scope The control lasts for the life of the bind session to allow for multiple write operations.

The use of the control implies that cryptographic consistency is verified by the caller. At bind time the presence of this control, along with the appropriate authorization, causes the `c_isConsistent` flag in the connection structure to be set to `TRUE`. This setting causes any write operations that contain pre-encrypted AES data to be accepted by the server. Without the presence of the control, the connection flag is set to `FALSE`, and a write operation of this type is rejected by the server. The RDBM back-end is the only back-end that sets, and evaluates, the `c_isConsistent` flag.

Audit control

The Audit control explains its use with the server and provides the results.

Description
The Audit Control enables a client to send more information about an operation. This additional information is a unique ID and an IP address. The additional information is audited in the audit log.

Note: This control is always enabled.

OID 1.3.18.0.2.10.22

Syntax

```
requestID DirectoryString

controlValue:=SEQUENCE {
{SEQUENCE of requestID}
clientIP  String
}
```

Behavior
This control is registered for the following operations:
- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

All users that include anonymous are enabled to send the control. However, there is an environment variable, `SLAPD_AUDIT_DISABLE_NON_ADMIN`, which when set, restricts the control to the following members:

- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members

If `SLAPD_AUDIT_DISABLE_NON_ADMIN` is set to `TRUE`, only audit controls that are sent by administrators are audited. By default the server enables any user to send this control.

Note: If non-admin users are disabled, and the control is sent by a non-admin, the control is ignored, even if it is critical.
If there is more information that is required for the control, the error is ignored, and the information is audited.

The Administration Server recognizes the control, but audits only one of these controls per operation. The behavior for the Administration Server is the same.

Scope The control lasts for the term of one operation. Each operation treats the control the same. If the operation is audited, the additional information that is sent in the control is audited as well.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: control ID
criticality: true | false
requestID: request ID sent in the control
requestID: request ID sent in the control
requestID: request ID sent in the control
clientIP: client IP sent in the control
```

Do not replicate control

The Do not replicate control explains its use with the server and provides the results.

Description
This control can be specified for an update operation. When present, a server does not replicate the update to any consumers.

OID 1.3.18.0.2.10.23

Syntax
This control has no value.

Behavior
This control is registered for the following operations:

Add When the control is detected in an add operation, the replication threads in a supplier does not replicate the add operation to the consumer.

Delete When the control is detected in a delete operation, the replication threads in a supplier does not replicate the delete operation to the consumer.

Modify
When the control is defected in a modify operation, the replication threads in a supplier does not replicate the modify operation to the consumer.

Modrdn
When the control is defected in a `modrdn` operation, the replication threads in a supplier does not replicate the modify operation to the consumer.

Any administrators and the Master Server DN are able to send the control.

The Administration Server does not support this control.

Scope The control lasts for the term of one operation.

Entry change notification control

The Entry change notification control explains its use with the server and provides the results.

Description
This control provides more information about the changes that caused a particular entry to be returned as the result of a persistent search.

OID 2.16.840.1.113730.3.4.7

Syntax

```
EntryChangeNotification ::= SEQUENCE {
                    changeType ENUMERATED {
                            add       (1),
                            delete    (2),
                            modify    (4),
                            modDN     (8)},
                    previousDN     LDAPDN OPTIONAL,
                    changeNumber   INTEGER OPTIONAL
}
```

Behavior
If the client set the `returnECs` Boolean to `TRUE` in the persistent search control, the server must include the entry change notification control in the controls portion of each `SearchResultEntry` that is returned because of an entry that is added, deleted, or modified.

The value of **changeType** field indicates what LDAP operation caused the entry to be returned.

The `previousDN` is present in `modifyDN` operations and is used to retrieve the DN of the entry before it was renamed or moved. The **changeType** optional field must be included by servers when it returns change notifications as a result of `modifyDN` operations.

The `changeNumber` field represents the change number `[CHANGELOG]` that is assigned by a server for the change. If a server supports an LDAP change log, it must include this field.

If the search code determines the persistent search control is present, the control is parsed and the operation is run as specified in the control. After the operation, the `pBlock` will be handed off to the plug-in for its record keeping, and the client search is left open. The `returnECs` control is returned from the plug-in and not the inline search code.

Note: It is up to the server administrator to configure change log for the client. If the change log is not set up properly, the client receives no change numbers.

This control is not supported by the Administration Server.

Group authorization control

The Group authorization control explains its use with the server and provides the results.

Description
The Group Authorization Control enables a bound user to assert group membership. The server uses this set of groups in the evaluation of ACLs for the operation. The control was introduced as a tool for the proxy server. However, this control can be sent by any client.

Note: This control is always enabled.

OID 1.3.18.0.2.10.21

Syntax

```
Group ::= SEQUENCE { groupName LDAPString }
RequestValue :: = SEQUENCE{
    normalized     INTEGER{unnormzlied(0), normalized(1)};
    Sequence of Group
}
```

The criticality must be set to `TRUE` to protect clients from submitting a request with an unauthorized identity.

Behavior
This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

The following persons are enabled to send the control:

- Primary Directory Administrator
- Proxy Authorization Group members
- Local Administration Group members
- Global Administration Group members

Only Primary Directory Administrator and Local Administration Group members can assert group membership into the global administration group. Proxy group members and global administration group members do not have the authority to assert group membership into the global administration group.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
If more information is required for the control, and if an error is in the formatting of that information, then the following error returns might occur:

- Missing information – `LDAP_OPERATIONS_ERROR`
- Additional information – `LDAP_OPERATIONS_ERROR`
- Invalid information – `LDAP_OPERATIONS_ERROR`

This control has the following possible return codes:

- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_OPERATIONS_ERROR`

This control is not supported by the Administration Server.

Scope The control lasts for the term of one operation. Each operation treats the control the same. The operation is run with the assumption that the user is a member of the stated groups. This operation applies to all back-end servers.

Auditing
This control has a special flag to indicate whether more information must be audited. If the audit flag `ibm-auditGroupsOnGroupControl` is set to `FALSE`, then the control OID and criticality are only audited. If `ibm-auditGroupsOnGroupControl` is `TRUE`, then the following information is audited:

```
controlType: control ID
criticality: {true | false}
Normalized: {true | false}
Group: group sent in request
Group: group sent in request
Group: group sent in request
```

LDAP delete operation timestamp control

The LDAP delete operation timestamp control explains its use with the server and provides the results.

Description
This control is used to send the modified timestamp values to a replica during a delete operation.

Note: This control is always enabled.

OID 1.3.18.0.2.10.32

Syntax

```
Control ::= SEQUENCE{
      controlType      1.3.18.0.2.10.32,
      criticality      BOOLEAN  FALSE,
      controlValue     OCTET STRING
}
```

where, the `OCTET STRING` value is a `BER` encoded value that represents the timestamp value of the delete operation.

Behavior

The LDAP delete operation timestamp control is registered for delete operations. This control contains the modified timestamp value for a replicated delete operation. It is used on a replica to update the corresponding value of a group entry, which undergoes group referential integrity check. This control does not rely on other controls and operates independently of other controls. The additional information is parsed to check for NULL value of timestamp for the control. If NULL is found, appropriate error code is returned.

Only the master server DN and supplier DN are authorized to send this control, since this control is used internally by the server replication code. If this control is sent by any other user, `LDAP_INSUFFICIENT_ACCESS` is returned.

The control has the possible return code as `LDAP_INSUFFICIENT_ACCESS`. This control is not supported by the Administration Server.

Scope This control lasts for one delete operation.

Limit number of attribute values control

The `Limit number of attribute values control` explains its use with the server and provides the results.

Description

This control limits the number of attribute values that are returned for an entry in a search operation. The `Limit number of attribute values control` is used to limit the number of values that are returned for the entire entry. This control can also be used to limit the number of values that are returned for attribute of an entry.

OID 1.3.18.0.2.10.30

Syntax

```
Control ::= SEQUENCE{
      controlType      1.3.18.0.2.10.30,
      criticality     BOOLEAN DEFAULT FALSE,
      controlValue    OCTET STRING OPTIONAL}
```

where, the OCTET STRING value is a BER encoded value with the following format:

```
RequestValue ::= SEQUENCE{
   MaxValuesPerEntry   INTEGER(0..maxInt),   // maximum number of values
                                             // for entire entry where
                                             // 0 means unlimited
   MaxValuesPerAttribute INTEGER(0..maxInt), // maximum number of values
                                             // per attribute where
                                             // 0 means unlimited
   ReturnDetails     BOOLEAN DEFAULT FALSE   // FALSE indicates that no
                                             // response controls should
                                             // be returned
}
```

The response that is sent with each entry whose attributes were partially returned when `ReturnDetails` is true is:

```
ResultValue ::= SEQUENCE{
   DN           LDAPString,       // The name of the attribute
                                  // in the same format returned
                                  // by search.
   AttributeList PartialAttributes // The list of partially returned
                                  // attributes for an entry.
}
```

where, `PartialAttributes` value is the BER encoded value with the following format:

```
  PartialAttributes ::= SEQUENCE of SEQUENCE{
    attributeName         LDAPString,        // The name of the attribute
                                             // in the same format as
                                             // returned by search.
    numberValuesReturned  INTEGER(0..maxInt),// number of values returned
                                             // for an attribute
    numberValuesAvailable INTEGER(-1..maxInt)// number of values available,
                                             // -1 if unknown
 }
```

Behavior

The `Limit number of attribute values control` is registered to be used along with the search operation. At a time, the control can be only used in one search operation. That is, the life of the control lasts only for a single search operation. All the users are authorized to use this control.

When the control is used in a search operation, a total number of attribute values are returned for each entry. The number is less than or equal to the maximum total number of values that are specified on the control. Also, the number of values that are returned per attribute is less than or equal to the maximum number of values that are returned per attribute. If details are requested on the control, a response control is also returned with each entry whose attributes were partially returned. This control is only supported by the RDBM back-end.

The `Limit number of attribute values control` operates independent of all other controls and does not affect the behavior of any other controls.

The following error codes might be returned if any additional information is required for this control and an error occurs in the formatting of that information:

- Missing information - `LDAP_DECODING_ERROR`
- Additional information - `LDAP_DECODING_ERROR`
- Invalid information - `LDAP_DECODING_ERROR`

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_DECODING_ERROR`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_NO_MEMORY`
- `LDAP_OTHER`
- `LDAP_UNWILLING_TO_PERFORM`

This control is not supported by the Administration Server.

Auditing

In this control, no additional information is audited.

Manage DSAIT control

The Manage DSAIT control explains its use with the server and provides the results.

Description

Causes entries with the `ref` attribute to be treated as normal entries, and allows clients to read and modify these entries.

OID 2.16.840.1.113730.3.4.2

Syntax

This control has no value.

Behavior
This control enables entries with the `ref` attribute to be treated as normal entries, and allows clients to read and modify these entries. In IBM Security Directory Server Proxy Server, without partitioned data, the request might not include this control. Even if the request does not include this control, the proxy server always sends the Manage DSAIT control to the back-end server. This control is registered for any operation. All users are enabled to send the control.

This control is not supported by the Administration Server.

Scope The control lasts for one operation.

Modify groups only control

The Modify groups only control explains its use with the server and provides the results.

Description
This control can be used with a delete, `modrdn`, or `moddn` operation to cause the server to modify the groups in which it is in a member without deleting or modifying the entry itself. The entry that is named in the delete, `modrdn`, or `moddn` request does not require to exist on the server.

Note: This control is always enabled.

OID 1.3.18.0.2.10.25

Syntax
This control has no value.

Behavior
This control is registered for the following operations:
- Delete
- Modrdn

The following persons are enabled to send the control:
- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
This control has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_DECODING_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`

The Administration Server does not support this control.

Scope The control lasts for the term of one operation. The control is only recognized when a delete, `moddn`, or `modrdn` request goes to the RDBM back-end.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: control ID
criticality: true | false
```

No replication conflict resolution control

The No replication conflict resolution control explains its use with the server and provides the results.

Description
When present, a replica server accepts a replicated entry without trying to resolve any replication conflict for this entry. This control can be used by the replication topology extended operation to ensure data consistency between a supplier and a consumer.

Note: If environment variable `IBMSLAPD_REPL_NO_CONFLICT_RESOLUTION` is set on a replica, a replica server acts as if all the update requests that coming from the suppliers are specified with this control. The replica accepts the replicated entries without attempting to resolve any replication conflicts. This environment variable is useful in a network topology in which one supplier and one or multiple consumers are defined.

OID 1.3.18.0.2.10.27

Syntax
This control has no value.

Behavior
This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn

Add Upon receiving such a control in a replicated Add request, a replica server does not try to resolve any replication conflict for this update. It accepts and applies it to the replica.

Modify
Upon receiving such a control in a replicated Modify request, a replica server does not try to resolve any replication conflict for this update. It accepts and applies it to the replica.

Only the Master Server DN is able to send the control.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

The Administration Server does not support this control.

Scope The control lasts for the term of one operation.

Omit group referential integrity control

The Omit group referential integrity control explains its use with the server and provides the results.

Description
This control enables an administrator to request that group referential integrity must not be run. The control applies only to `modrdn` and delete operations. When present on a delete or rename operation, the entry is deleted from or renamed in the directory. But the entry membership is not removed or renamed in the groups in which the entry is a member.

Note: This control is always enabled.

OID 1.3.18.0.2.10.26

Syntax
This control has no value.

Behavior
This control is registered for the following operations:
- Delete
- Modrdn

The following persons are enabled to send the control:
- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
This control has the following possible return codes:
- `LDAP_SUCCESS`
- `LDAP_DECODING_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`

The Administration Server does not support this control.

Scope The control lasts for the term of one operation. The control is only recognized when a delete, `moddn`, or `modrdn` request goes to the RDBM back-end.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:
```
controlType: control ID
criticality: true | false
```

Paged search results control

The Paged search results control explains its use with the server and provides the results.

Description
The paged results control is enabled on a search operation and enables a client to request a subset of entries. Subsequent search requests for using this control continue to result in the next results page until the operation is canceled or the last result is returned. This control is supported by RDBM back-end and by Proxy server version 6.2 and later.

Note: This control can be disabled by setting the Paged result limit to `0`.

There is also a configuration option which enables an administrator to grant or deny the use of this control to non-administrators. The administrators in this case refer to the primary directory administrator, local administration group members, and global administration group members. If the **`ibm-slapdPagedResAllowNonAdmin`** attribute in the `cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration` entry is set to `TRUE`, all users can send paged search requests. If set to `FALSE`, only administrators can send paged search requests against RDBM back-end.

In a proxy server, if **ibm-slapdPagedResAllowNonAdmin** is set to FALSE, then only Global Administration Group members are allowed to do page search. If primary directory administrator or local administration group members runs page search when the attribute is set to FALSE, then LDAP_INSUFFICIENT_ACCESS is retuned.

OID 1.2.840.113556.1.4.319

Syntax

```
realSearchControlValue ::= SEQUENCE {
 Size  INTEGER(0..maxInt),
    -- requested page size from client
    -- result set size estimate from server
 Cookie  OCTET STRING }
```

Behavior

This control is registered for the following operations:

- Search

In a default user installation, any user can send this control. If the **ibm-slapdSortSrchAllowNonAdmin** is set to FALSE, the use of this control is restricted to administrative users:

- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.
If more information is required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – LDAP_DECODING_ERROR
- Additional information – LDAP_DECODING_ERROR
- Invalid information – LDAP_DECODING_ERROR

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_DECODING_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_OTHER

The Administration Server does not support this control.

Scope The control lasts for the term of one operation. The control changes the behavior of a search operation that goes against the RDBM back-end. The control requests that the server return the entries in a sorted order. The configuration back-end and schema back-ends do no support this control.

Auditing

When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: control ID
criticality: true | false
```

Password policy request control

The Password policy request control explains its use with the server and provides the results.

Description

This control is sent by the client application with the requested operation. This control indicates to the server that this client understands Password Policy return values. If the client sends the Password policy request control with the request, the server can send the Password policy request control with the response. The Password policy request control contains extra information about why an operation failed because of a Password Policy problem. For example, if a client bind request failed because the user account is locked out. This information is sent to the client on the response in the **Password Policy Response Control** value field.

Note: If the Password Policy is disabled, then the Password policy request control is ignored, so no Password policy request control is sent with the response.

Request

OID 1.3.6.1.4.1.42.2.27.8.5.1

Syntax

There is no request value for the control.

Response

OID 1.3.6.1.4.1.42.2.27.8.5.1

Syntax

```
SEQUENCE {
      warning   [0] CHOICE OPTIONAL {
         timeBeforeExpiration  [0] INTEGER (0 .. MaxInt),
         graceLoginsRemaining  [1] INTEGER (0 .. maxInt) }
      error     [1] ENUMERATED OPTIONAL {
         passwordExpired       (0),
         accountLocked         (1),
         changeAfterReset      (2),
         passwordModNotAllowed (3),
         mustSupplyOldPassword (4),
         invalidPasswordSyntax (5),
         passwordTooShort      (6),
         passwordTooYoung      (7),
         passwordInHistory     (8) } }
```

Behavior

This control is registered for the following operations:

- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

All users are enabled to send the control.This control has the following possible return codes:

- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_INVALID_CREDENTIALS`
- `LDAP_CONSTRAINT_VIOLATION`

- `LDAP_UNWILLING_TO_PERFORM`

The Administration Server supports this control. The Administration Server checks for this control on the bind operation, and returns the Password policy response control and values if needed. If the Root Administrator has too many bad binds in a row, the Administration Server locks out the account. It sends the Password Policy response that the account is locked.

Scope The control lasts for the term of one operation. This control indicates to the server that the client application has information about Password Policy. Therefore, the server sends a Password policy response control with its response. With this response control, there can be a response control value which contains the Password Policy error or warning code and message if one is required. The other back-ends have no knowledge of this control and so it is ignored.

Persistent search control

The Persistent search control explains its use with the server and provides the results.

Description
This control provides clients a means to receive notification of changes in the LDAP server.

OID 2.16.840.1.113730.3.4.3

Syntax

```
PersistentSearch ::=  SEQUENCE {
         changeTypes      INTEGER,
         changesOnly      BOOLEAN,
         returnECs        BOOLEAN}
```

Behavior
This control can be used by all LDAP users.

If **changesOnly** is `TRUE`, then the server does not return any existing entries that match the search criteria. Also, no entries are returned until an update on an entry occurs that matches the initial search filter. Entries are only returned after successful update operations. For example, if data is loaded in the server and a search is issued against it, the matching entries are returned. However, if the persistent search control is present the entries might or might not be returned initially. This search is determined by the **changesOnly** field.

If **changesOnly** is `FALSE`, then the server returns all the entries that match the search filter. Also, the connection is left open and any changes or updates on entries that match the search filter from that point triggers entries to be returned.

The **changeTypes** is the logical OR of one or more of these values:

- add (1)
- delete (2)
- modify (4)
- modDN (8)

If **returnECs** is `TRUE`, the server returns an entry change notification control with each entry returned as the result of changes.

This control is not supported by the Administration Server.

Proxy authorization control

The Proxy authorization control explains its use with the server and provides the results.

Description

The Proxy Authorization Control enables a bound user to assert another user identity. The server uses this asserted identity in the evaluation of ACLs for the operation.

Note: This extended operation is always enabled.

OID 2.16.840.1.113730.3.4.18

Syntax

User DN can be one of the following values:

```
dn: dn value
dn value
RequestValue:: = User DN
```

Behavior

This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

The following persons are enabled to send the control:

- Primary Directory Administrator
- Proxy Authorization Group members
- Local Administration Group members
- Global Administration Group members

No user can assert the identity of the primary directory administrator or local administration group members. Only a primary directory administrator or local administration group members can assert the identity of a global administration group member. Global administration group members and proxy group members cannot assert the identity of a global administration group member.

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.
If more information is required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – `LDAP_OPERATIONS_ERROR`
- Additional information – `LDAP_OPERATIONS_ERROR`
- Invalid information – `LDAP_OPERATIONS_ERROR`

This control has the following possible return codes:

- `LDAP_SUCCESS`

- `LDAP_OPERATIONS_ERROR`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_OTHER`
- `LDAP_NO_MEMORY`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_PARAM_ERROR`

This control is not supported by the Administration Server.

Scope The control lasts for the term of one operation. Each operation treats the control the same. The operation is run after you assume the asserted user identity. The control is recognized on all operations.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:

`ProxyDN:` *`proxy dn`*

Refresh entry control

The Refresh entry control explains its use with the server and provides the results.

Description
This control is returned to a supplier when a consumer server detects a replication conflict during a replicated modify operation. Upon receiving such a control along with an `LDAP_OTHER` return code, the supplier retrieves its copy of the entry and send the entry again to the consumer by using an add operation to refresh the consumer version of the entry.

OID 1.3.18.0.2.10.24

Syntax
This control has no value.

Behavior
This control is registered for the following operations:

- Modify

This control is sent in an LDAP response protocol after a conflict is detected on a replicated entry on a consumer. The consumer does not require to specifically bind to the supplier to return such a control. The supplier is already bound to the consumer. If anybody sends such a control in an LDAP request to any server, the control is ignored and has no effect on the server.

The Administration Server does not support this control.

Scope The control lasts for the term of one operation. This control is used by a consumer to communicate to its supplier when a replication conflict is detected on the consumer. When the supplier gets along the control with an `LDAP_OTHER` return code, the supplier sends the entry again with an intention of bringing the consumer back in sync.

Replication bind failure timestamp control

The master server uses the replication bind failure timestamp control to propagate the bind failure timestamp value to a read-only replica server.

Description

A master server in a replication topology supports this control only when the `ibm-replicateSecurityAttribute` attribute is set to `true`.

A read-only replica notifies its master when a user attempts a bind operation against it that results in password policy operational attributes update. The read-only replica server notifies its master server only if the following conditions are met:

- The `ibm-replicateSecurityAttribute` attribute is set on a read-only replica server.
- The `ibm-replicareferralURL` attribute is set with the IP address or fully qualified domain name with ports of all its master servers.

The read-only replica server notifies its master server and provides the following values:

- Passes the replication bind failure timestamp control.
- Binds with the user credentials that resulted in password policy operational attributes update.

If the `ibm-replicateSecurityAttribute` attribute is set on the master server, it propagates bind failure timestamp value to the read-only replica in its response. The read-only replica records the failure timestamp in its database for the user entry. Therefore, both the read-only replica and master server record the same password failure timestamp for the bind.

If the `ibm-replicateSecurityAttribute` attribute is not set on the master server, the master server does not interpret the control that it receives from the read-only replica. If the master is unable to interpret the control, it does not return the password failure timestamp in its response to the read-only replica. However, master server updates its password failure count and then replicates it to other servers. If the read-only replica does not receive the password failure timestamp from the master, it records the password failure timestamp at its end in the database.

OID `1.3.18.0.2.10.34`

Syntax

The master server sets the response value with the bind failure timestamp in the string format.

Behavior

This control is registered only for a bind operation.

A read-only replica server sends the control to an identified master as an internal request along with the bind operation by using the same user credentials. The read-only replica sends the request when a user attempts a bind operation that results in invalid credentials, password expiration, or password grace use time. In the control response, the master server sends a timestamp value of invalid credentials, password expiration, or password grace use time to the read-only replica. Only a master server supports this control and is meant for an RDBM back-end server.

Any user can send this control. A master server expects this control from a read-only replica. If any external client or user sends this control, the master server functions returns a password failure timestamp. It is a non-critical control and return code does not vary.

No additional information is required for the control. The master server ignores any additional information that is receives. The return response from the master server might contain a password failure timestamp value.

A read-only replica uses the `ldap_get_result_control` call to parse the response for the timestamp. If there is an error in the format of the timestamp, the read-only replica takes the following actions:

- Ignores the timestamp from the master server.
- Records the timestamp of the read-only replica in the user entry.

This control operates independent of other controls. Therefore, the control does not depend on other control and does not affect the behavior of other controls.

This control is not supported by the Administration Server.

Scope The control lasts for one operation.

Auditing
In this control, the following information is audited:

- OID
- Timestamp in string format

Replication supplier bind control

The Replication supplier bind control explains its use with the server and provides the results.

Description
Gateway servers send only the changes they receive from a gateway to their local servers. The servers are in the same site as the gateway server, including peer, forwarder, or pelican server. They do not send these changes to the other gateway servers. The Replication supplier bind control helps a gateway server to decide which servers to send to and what to send them. When a gateway server binds to its consumers, it sends the control with its `serverID` as the control value. When a gateway server receives such a control in a bind request, it knows that a gateway server is bound as a supplier.

OID 1.3.18.0.2.10.18

Syntax

```
controlValue :: SEQUENCE {
 SupplierServerId   LDAPString
}
```

Behavior
This control is registered for the following operations:

- Bind

Only the Master DN is enabled to send this control.

Note: If the control is sent by a user who does not have access, `LDAP_UNWILLING_TO_PERFORM` is returned.
If more information is required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – `LDAP_OPERATIONS_ERROR`
- Additional information – ignored
- Invalid information – ignored

This control has the following possible return codes:

- `LDAP_PROTOCOL_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`

The Administration Server does not support this control.

Scope The control lasts for the life of the bind session. When the control is received, a server knows that a gateway server is bound as a supplier. Depending on the supplier information, the server can decide to which consumers an entry is to be replicated.

Replication update ID control

The Replication update ID control explains its use with the server and provides the results.

Description
This control was created for serviceability. If the supplier server is set to issue the control, each replicated update is accompanied by this control. The data in this control can be used to identify problems with multi-threaded replication and replication conflict resolution. By default, no supplier includes this control.

Note: This control is always enabled.

OID 1.3.18.0.2.10.29

Syntax

```
replication agreement DN:replication change ID
```

These values are set by the supplier.

Behavior
This control is not registered by any operations.

All users are enabled to send the control.

The Administration Server does not support this control.

Scope The control lasts for one operation.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: OID
criticality: false
value: Replication agreement DN:change ID
```

Return deleted objects control

The Return deleted objects control explains its use with the server and provides the results.

Description
The return deleted objects control when included in a null base search request, all entries in the database and those entries with attribute `isDeleted` set to `TRUE` are returned.

OID 1.3.18.0.2.10.33

Syntax
This control has no value.

Behavior

When this control is included in a null base search request, all entries in the database and those entries with attribute `isDeleted` set to `TRUE` are returned. Normally, all the entries under "cn= Deleted Objects" have the attribute isDeleted set to TRUE, and entries under other subtree do not have this attribute defined.

This control applies only to a server with RDBM back-end. When this control is included with search base, cn=Deleted Objects, all the entries under the subtree are returned, even those entries with attribute isDeleted not set to TRUE. When the control is included with search base other than cn=Deleted Objects, no entries are returned.

Users with sufficient ACL to access the cn=Deleted Objects subtree are able to search and update the entries under the subtree on a directory server instance. The users include primary administrator, local administration group members with DirDataAdmin role, and global administrator group members. Local administrator group members with Server Configuration group member rights (ServerConfigGroupMember) are able to update the attributes of the entries that are related to tombstone in the configuration file.

If the control is sent by users who do not have sufficient access permissions, then LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_NO_SUCH_OBJECT
- LDAP_NO_MEMORY
- LDAP_INSUFFICIENT_ACCESS
- LDAP_OPERATIONS_ERROR

The Administration Server does not support this control.

Scope This control lasts for one search operation.

Server administration control

The Server administration control explains its use with the server and provides the results.

Description

Allows an update operation by the administrator under conditions when the operation is normally refused. For example, the server is quiesced, the server is a read-only replica, and others).

This control can be specified on an update operation (add, modify, modRdn, modDn, delete) by a client that is bound as an administrator. This control can also be specified on a bind-related operation. On a bind operation, this control specifies that it is an administrative connection and the connection must not be dropped when the idle connections are cleaned. This control is only recognized if a client is bound as a primary directory administrator, global admin group member, or member of the administrative group with any role other than the "NoAdmin" role. When present, a server that would normally refuse updates (quiesced server, forwarder, or replica), allows the update. The updates are replicated like other updates.

Note: This control requires to be used with user discretion. With the control, entry updates are allowed under unusual circumstances. Therefore, it is the responsibility of the user to ensure the server that is updated ends up in a state consistent with the other servers. For example, the timestamp of an entry which is used as the base for replication conflict resolution in

IBM Security Directory Server, version 6.0 and later. The timestamp might be different on different servers if the entry gets updated individually on those servers with this control.

OID 1.3.18.0.2.10.15

Syntax
This control has no value.

Behavior
This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn
- Moddn
- Bind
- Unbind
- Search

Administrator Server supports the following extended operations:

- Attribute type
- DN normalization
- Dynamic update requests
- Get lines
- Get number of lines
- LDAP trace facility
- LogMgmtControl
- ServerBackupRestore
- Start, stop server
- Start TLS
- User type

Administrator Server supports the following controls:

- Audit
- Password policy request
- Server administration

The following persons are enabled to send the control:

- Primary Directory Administrator
- Local Administration Group Member
- Global Administration Group Member

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

Scope The control lasts for one operation. When the control is received, a server knows that a gateway server is bound as a supplier. Depending on the supplier information, the server can decide to which consumers an entry is to be replicated.

Sorted search results control

The Sorted search results control explains its use with the server and provides the results.

Description

The sorted search results control enables a client to receive search results that are sorted by a sort key.

Note: This control can be disabled by setting the **ibm-slapdSortKeyLimit** to 0.

There is also a configuration option which enables an administrator to grant or deny the use of this control to non-administrators. In this case, administrators refer to the primary directory administrator, local administration group members, and global administration group members. If the **ibm-slapdSortSrchAllowNonAdmin** attribute in the cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration entry is set to TRUE, then all users are enabled to use the sorted search. If set to FALSE, only administrators can use the sorted search.

OID 1.2.840.113556.1.4.473

Syntax

```
The controlValue is an OCTET STRING who value is the
BER encoding of a value with the following SEQUENCE:

SortKeyList ::= SEQUENCE of SEQUENCE {
 AttributeType  AttributeDescription,
 OrderingRule [0] MatchingRuleId OPTIONAL,
 ReverseOrder [1] BOOLEAN DEFAULT FALSE }
```

Behavior

This control is registered for the following operations:
- Search

In a default user installation, any user can send this control. If the **ibm-slapdSortSrchAllowNonAdmin** is set to FALSE, the use of this control is restricted to administrative users:
- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.
If more information is required for the control, and there is an error in the formatting of that information, the following error returns might occur:
- Missing information – LDAP_DECODING_ERROR
- Additional information – LDAP_DECODING_ERROR
- Invalid information – LDAP_DECODING_ERROR

This control has the following possible return codes:
- LDAP_SUCCESS
- LDAP_DECODING_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_OTHER

The Administration Server does not support this control.

Scope The control lasts for the term of one operation. The control changes the behavior of a search operation that goes against the RDBM back-end. The control requests that the server return the entries in a sorted order. The configuration back-end and schema back-ends do no support this control.

Auditing
When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: control ID
criticality: true | false
```

Subtree delete control

The Subtree delete control explains its use with the server and provides the results.

Description
This control is attached to a delete request. This control indicates that the specified entry and all descendant entries are to be deleted. However, if the subtree is an active replication context, the control does not take effect and an `LDAP_UNWILLING_TO_PERFORM` message is returned. This return means that the subtree to be deleted might contain any replication agreements that the server uses to replicate. If the subtree contains any replication agreements, then the subtree cannot be deleted by using this control.

OID 1.2.840.113556.1.4.805

Syntax
This control has no value.

Behavior
This control is registered for the following operations:

- Delete

The following persons are enabled to send the control:

- Primary Directory Administrator
- Local Administration Group members
- Global Administration Group members
- Master server DN

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

This control has the following possible return codes:

- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_UNWILLING_TO_PERFORM`

The Administration Server does not support this control.

Scope The control lasts for the term of one delete operation. The delete operation not only deletes the base entry that is specified in the request, but also deletes all the descendant entries.

Transaction control

The Transaction control explains its use with the server and provides the results.

Description
The Transaction control is sent along with update operations run within a transaction.

Note: This control is enabled by default, but can be disabled by changing the value in the configuration file for the **ibm-slapdTransactionEnable** attribute.

The **ibm-slapdTransactoinEnabled** attribute is in the configuration file in the cn=Transaction,cn=configuration entry. If the value is set to FALSE, transactions are not enabled. If set to TRUE, transactions are enabled. Transactions can also be enabled or disabled by using the web administration tool.

OID 1.3.18.0.2.10.5

Syntax

The controlValue is set to the transaction ID returned in the StartTransaction response.

Behavior

This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn

Any user can send this control.

If more information is required for the control, and the transaction ID sent in the control does not match the transaction ID on the connection, then LDAP_PROTOCOL_ERROR is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_TIMELIMIT_EXCEEDED
- LDAP_SIZELIMIT_EXCEEDED

The Administration Server does not support this control.

Scope The control lasts for the term of one operation, but must be sent only in a transactional context. When the control is sent only with an update operation to the RDBM back-end, the server holds the update until an end-transaction request is received. The control is only supported on updated operations that are run in a transactional context (a start transaction extended operation must be run first).

Auditing

When the server receives this control, the audit plug-in adds the following lines to the audit entry:

```
controlType: control ID
criticality: true | false
```

Virtual list view control

The Virtual list view control explains its use with the server and provides the results.

Description

The Virtual list view control extends the regular LDAP search operation and includes a server-side sorting control. With this control, the server returns a contiguous subset of entries that are taken from an ordered result set that match a criteria of a target entry, rather than returning the complete set of `searchResultEntry` messages.

Request

OID 2.16.840.1.113730.3.4.9

Syntax

```
VirtualListViewRequest ::= SEQUENCE {
     beforeCount   INTEGER (0..maxInt),
     afterCount    INTEGER (0..maxInt),
     target      CHOICE {
              byOffset       [0] SEQUENCE {
                   offset          INTEGER (1 .. maxInt),
                   contentCount    INTEGER (0 .. maxInt) },
              greaterThanOrEqual [1] AssertionValue },
     contextID OCTET STRING OPTIONAL }
```

where,

- `beforeCount` - Indicates the number of entries before the target entry that the client wants the server to send.
- `afterCount` - Indicates the number of entries after the target entry that the client wants the server to send.
- `offset` - Used to identify the target entry in the Virtual list view request control by determining the target entry that is offset within the list. The server examines the `contentCount` and offset given by the client and computes the corresponding offset within the list by using following formula:

  ```
  Si = Sc * (Ci / Cc)
  ```

  where,
  - `Si` is the actual list offset that is used by the server
  - `Sc` is the server estimate for content count
  - `Ci` is the client submitted offset
  - `Cc` is the client submitted content count
- `contentCount` - Used to identify the target entry.
- `greaterThanOrEqual` - Indicates a matching rule assertion value. If present, its value is used to determine the target entry by comparing with the attribute values specified as the primary sort key.
- `contextID` - Contains a value for the most recently received **contextID** field for the same list view from the Virtual list view response control.

Response

OID 2.16.840.1.113730.3.4.10

Syntax

```
VirtualListViewResponse ::= SEQUENCE {
      targetPosition INTEGER (0 .. maxInt),
      contentCount INTEGER (0 .. maxInt),
      virtualListViewResult ENUMERATED {
           success (0),
           operationsError (1),
           protocolError (2),
           unwillingToPerform (53),
           insufficientAccessRights (50),
           timeLimitExceeded (3),
```

```
          adminLimitExceeded (11),
          innapropriateMatching (18),
          sortControlMissing (60),
          offsetRangeError (61),
          other(80),
          ... },
     contextID OCTET STRING OPTIONAL }
```

where,

- `targetPosition` - Indicates the offset list for the target entry.
- `contentCount` - Indicates the number of entries in the list that is based on the server estimate. The value of the count depends on the access rights over all the entries for the user who is bound to the directory server.
- `contextID` - Server defined octet string.
- `virtualListViewResult` - Contains error messages that are related to the Virtual list view operation. For example, `insufficientAccessRights` indicates that the server denies the client the permission to run the Virtual list view operation.

Behavior

The Virtual list view control extends the regular LDAP Search operation and includes a server-side sorting control. In this operation, the server returns a contiguous subset of entries that are taken from an ordered result set that match a criteria of a target entry, rather than returning the complete set of `searchResultEntry` messages.

When you send this control, it must have an accompanying server-side sorting control. If server-side sorting control is not specified, the request is rejected with the `LDAP_SORT_CONTROL_MISSING` error.

IBM Security Directory Server, version 6.2 or later server versions recognize the Virtual list view request control that is sent along with a search request. The search request is then passed to the RDBM back-end from which the result set is fetched. Then, the cursor is positioned at the required offset in the search result set.

All users are authorized to use this control. This control can be enabled or disabled on a server by using a configuration option.

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_PROTOCOL_ERROR`
- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_TIMELIMIT_EXCEEDED`
- `LDAP_ADMINLIMIT_EXCEEDED`
- `LDAP_INAPPROPRIATE_MATCHING`
- `LDAP_SORT_CONTROL_MISSING`
- `LDAP_INDEX_RANGE_ERROR`
- `LDAP_OTHER`

The Administrator Server does not support this control.

Scope This control lasts only for one search operation.

Appendix G. Client libraries

The 32-bit and the 64-bit libraries have the same names.

The following table lists the libraries that are built for IBM Tivoli Directory Server, version 6.0 and later as part of client.

Table 14. Supported libraries on different platforms

Libraries	**Operating Systems**				
	AIX	**HPUX**	**Linux**	**Solaris**	**Windows for IA32**
`idsldap_` `plugin_` `ibm_gsskrb`	Y	NA	NA	NA	NA
`idsldap_` `plugin_` `sasl_` `digest-md5`	Y	Y	Y	Y	Y
`libidsldap`	Y	Y	Y	Y	Y
`libidsldapn`	Y	NA	NA	NA	Y
`libids` `ldapstatic`	Y	Y	Y	Y	Y
`libids` `ldapstaticn`	Y	NA	NA	NA	Y
`libids` `ldapiconv`	Y	Y	Y	Y	Y
`libidsldif`	NA	Y	Y	NA	NA
`libids` `ldifstatic`	Y	Y	Y	Y	Y
`libibm` `ldapdbg`	Y	Y	Y	Y	Y
`ldap`	NA	NA	NA	NA	Y
`ldapstatic`	NA	NA	NA	NA	Y

Note: The dynamic version of `libldif` is available on Linux, but not on Solaris.

Legend:

Y	This library is 64-bit recertified on the corresponding operating system.

NA This library is not 64-bit recertified, or it is not valid for the corresponding operating system.

Hence the architecture (32-bit or 64-bit) that is used for those binary files is the one that is used for these libraries, as well. Therefore, these libraries are placed in the appropriate folder (`lib` or `lib64`).

Note: The following library extensions are applicable for each platform:

Table 15. Library extensions on different platforms

Platform	Static library	Shared (Dynamic) library
AIX	.a	.a
Linux	.a	.so
Solaris	.a	.so
Windows	.lib	.dll

Appendix H. Sample Makefile

The sample Makefile (makefile.ex) is updated with the rules and information about building 64-bit clients.

The sample Makefile lists the 64-bit compilers or linkers to be used along with the relevant flags to be passed. It also lists the 64-bit libraries, needed to build the customized LDAP clients.

The following sample shows the makefile for 64-bit Linux:

```
#-----------------------------------------------------------------------
#
# ABSTRACT: makefile to generate the example LDAP client programs
#
# Licensed Materials - Property of IBM
#
# 5724-J39
#
# (C) Copyright IBM Corp. 1997, 2007 All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#-----------------------------------------------------------------------
# Copyright (c) 1994 Regents of the University of Michigan.
# All rights reserved.
#
# Redistribution and use in source and binary forms are permitted
# provided that this notice is preserved and that due credit is given
# to the University of Michigan at Ann Arbor. The name of the University
# may not be used to endorse or promote products derived from this
# software without specific prior written permission. This software
# is provided ``as is'' without express or implied warranty.
#-----------------------------------------------------------------------
#
# This makefile will build the example programs whose source is contained
# in this directory.  The four programs generated are:
#  ldapsearch
#  ldapmodify
#  ldapadd (a hard-link to ldapmodify)
#  ldapmodrdn
#  ldapdelete
#  ldapchangepwd
#  ldapexop
# In addition to being the examples of  use of the LDAP client api, these
# programs are useful command line utilities. See the README file for
# more details.

#
# default definitions for Unix utilities (may be changed here)
CC = gcc
RM = rm -f
HARDLN = ln
MKDIR = mkdir -p

# The following variable indicates the architecture of the output binaries
# on using this Makefile.

BITS = 64

###########################################################################
```

```
## General compiler options                                               ##
############################################################################

DEFINES = -DLINUX -D_GCC3
#Note: Append the path to appropriate LDAP headers if not already present
#in the include list.
INCLUDES = -I/opt/ibm/ldap/V6.3.1/include -I../include -I/usr/include

############################################################################
## Options for building 32-bit targets on AMD64 Linux
############################################################################
#-------------------------------------------------------------------
# Use the following definition to link the sample programs with
# the shared LDAP library dynamically.
# CLIENT_LIBS = -lidsldif -libmldap -libmldapdbg -lidsldapiconv
# LIBS = -L/opt/ibm/ldap/V6.3.1/lib -L/usr/lib -lpthread -ldl
#-------------------------------------------------------------------
# Or use this definition to link the LDAP library statically:
# CLIENT_LIBS = -libmldapstatic -lidsldifstatic
# LIBS = -L/opt/ibm/ldap/V6.3.1/lib -L../lib -L/usr/lib -lpthread -ldl
#-------------------------------------------------------------------
# LFLAGS = -Wl,-rpath,/opt/ibm/ldap/V6.3.1/lib $(LIBS) $(CLIENT_LIBS)
# CFLAGS = $(INCLUDES) $(DEFINES) -m32
############################################################################
## Options for building 64 bit targets on AMD64 Linux
############################################################################
#------------------------------------------------------------------
# Use the following definition to link the sample programs with
# the shared LDAP library dynamically
 CLIENT_LIBS = -lidsldif -libmldap -libmldapdbg -lidsldapiconv
 LIBS = -L/opt/ibm/ldap/V6.3.1/lib64 -L/usr/lib64 -lpthread -ldl
#------------------------------------------------------------------
# Or use this definition to link the LDAP library statically
# CLIENT_LIBS = -libmldapstatic -lidsldifstatic
# LIBS = -L/opt/ibm/ldap/V6.3.1/lib64 -L../lib64 -L/usr/lib64 -lpthread -ldl
#------------------------------------------------------------------
 LFLAGS = -Wl,-rpath,/opt/ibm/ldap/V6.3.1/lib64 $(LIBS) $(CLIENT_LIBS)
 CFLAGS = $(INCLUDES) $(DEFINES)
############################################################################
## Targets                                                                ##
############################################################################

all: ldapsearch ldapmodify ldapdelete ldapmodrdn ldapadd ldapchangepwd ldapexop

ldapsearch:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapsearch.c $(LFLAGS)

ldapmodify:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapmodify.c $(LFLAGS)

ldapdelete:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapdelete.c $(LFLAGS)

ldapmodrdn:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapmodrdn.c $(LFLAGS)

ldapchangepwd:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapchangepwd.c $(LFLAGS)

ldapexop:
 $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) -o $(BITS)/$@ ldapexop.c $(LFLAGS)
```

```
ldapadd: ldapmodify
 $(RM) $(BITS)/$@
 $(HARDLN) $(BITS)/ldapmodify $(BITS)/ldapadd

clean:
 $(RM) *.o core a.out $(BITS)/*.o $(BITS)/core $(BITS)/a.out $(BITS)/ldapsearch \
      $(BITS)/ldapmodify $(BITS)/ldapdelete \
  $(BITS)/ldapmodrdn $(BITS)/ldapadd $(BITS)/ldapchangepwd $(BITS)/ldapexop
```

The following example shows a sample `makefile` for 64-bit Windows operating system:

```
#
#
#----------------------------------------------------------------------
#
# ABSTRACT: makefile to generate the example LDAP client programs
#
# Licensed Materials - Property of IBM
#
# 5724-J39
#
# (C) Copyright IBM Corp. 1997, 2007 All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#----------------------------------------------------------------------
# Copyright (c) 1994 Regents of the University of Michigan.
# All rights reserved.
#
# Redistribution and use in source and binary forms are permitted
# provided that this notice is preserved and that due credit is given
# to the University of Michigan at Ann Arbor. The name of the University
# may not be used to endorse or promote products derived from this
# software without specific prior written permission. This software
# is provided ``as is'' without express or implied warranty.
#----------------------------------------------------------------------
#

# This makefile will build the example programs whose source is contained
# in this directory. The four programs generated are:
#  ldapsearch
#  ldapmodify
#  ldapadd (a hard-link to ldapmodify)
#  ldapmodrdn
#  ldapdelete
#  ldapchangepwd
#  ldapexop
# In addition to being the examples of use of the LDAP client api, these
# programs are useful command line utilities. See the LDAP Programming
# Reference for more details.

#
# default definitions for client utilities (may be changed here)
CC = $(SDKROOT)\bin\win64\x86\AMD64\cl.exe
LD = $(SDKROOT)\bin\win64\x86\AMD64\link.exe
RM = del /f
HARDLN = copy
MKDIR = md

# Specify the directory where the C SDK is installed. In our case the Feb 2003
# version of the VC SDK was used.
SDKROOT = C:\sdk\ms\2003_sp1

# Specify the directory where ISDS 6.3.1 clients are installed
```

```
INSTALLROOT = C:\Progra~1\IBM\ldap\V6.3.1

# The following variable indicates the architecture of the output binaries
# on using this Makefile.

BITS = 64

##############################################################################
## General compiler options                                                 ##
##############################################################################

DEFINES = /DNDEBUG /DWIN32 /D_CONSOLE /D_MBCS /DNT
INCLUDES = /I$(INSTALLROOT)\include /I../include /I$(SDKROOT)\include\crt
           /I$(SDKROOT)\include\crt\sys /I$(SDKROOT)\include
CFLAGS = /nologo /MD /EHsc /Od $(INCLUDES) $(DEFINES) /Fo$(BITS)/

##############################################################################
## Options for building 64-bit targets for Windows (AMD64)                   ##
##############################################################################
LIBS = kernel32.lib uuid.lib msvcrt.lib oldnames.lib Wsock32.lib
       AdvAPI32.lib bufferoverflowu.lib
#------------------------------------------------------------------------
# Use the following definition to link the sample programs with
# the LDAP shared library.
CLIENT_LIBS = libidsldap.lib libibmldapdbg.lib libidsldifstatic.lib
LDIR = /LIBPATH:$(INSTALLROOT)\lib64 /LIBPATH:$(SDKROOT)\Lib\AMD64
#------------------------------------------------------------------------
# Use the following definition to link the sample programs statically.
#CLIENT_LIBS = libibmldapdbgstatic.lib libidsldapstatic.lib libidsldifstatic.lib
#LDIR = /LIBPATH:$(INSTALLROOT)\lib64 /LIBPATH:..\lib64 /LIBPATH:$(SDKROOT)\Lib\AMD64
#------------------------------------------------------------------------
LFLAGS = /nologo /subsystem:console /incremental:no \
 $(LDIR) $(LIBS) $(CLIENT_LIBS)
# Note : In case the libraries aren't picked up using the above syntax, modify
# the env. variable PATH & LIB to point to the path of the requisite libraries.
#
##############################################################################
## Targets                                                                  ##
##############################################################################

all: ldapsearch.exe ldapmodify.exe ldapdelete.exe ldapmodrdn.exe
             ldapadd.exe ldapchangepwd.exe ldapexop.exe

ldapsearch.exe: $(BITS)/ldapsearch.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapsearch.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapmodify.exe: $(BITS)/ldapmodify.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapmodify.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapdelete.exe: $(BITS)/ldapdelete.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapdelete.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapmodrdn.exe: $(BITS)/ldapmodrdn.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapmodrdn.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapchangepwd.exe: $(BITS)/ldapchangepwd.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapchangepwd.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapexop.exe: $(BITS)/ldapexop.obj $(BITS)/getopt.obj
 -@ $(MKDIR) $(BITS)
 $(CC) $(CFLAGS) /c getopt.c
 $(CC) $(CFLAGS) /c ldapexop.c
 $(LD) $(LFLAGS) /out:$(BITS)/$@ $**
 $(RM) $(BITS)\*.obj $(BITS)\*.exp $(BITS)\*.lib

ldapadd.exe: ldapmodify.exe
 -@ del /f $(BITS)/$@
 $(HARDLN) $(BITS)\ldapmodify.exe $(BITS)\ldapadd.exe

clean:
 $(RM) *.obj *.exe *.exp *.lib $(BITS)\*.obj $(BITS)\*.exe $(BITS)\*.exp
         $(BITS)\*.lib

$(BITS)/getopt.obj: getopt.c

$(BITS)/ldapsearch.obj: ldapsearch.c

$(BITS)/ldapmodify.obj: ldapmodify.c

$(BITS)/ldapdelete.obj: ldapdelete.c

$(BITS)/ldapmodrdn.obj: ldapmodrdn.c

$(BITS)/ldapchangepwd.obj: ldapchangepwd.c

$(BITS)/ldapexop.obj: ldapexop.c
```

You can find the sample `Makefile` (`makefile.ex`) in *`ldap_home`*`/examples`.

Appendix I. Limited transaction support

The limited transaction support provides information about its properties.

Transactions have four critical properties:

atomicity
: The transaction must be run completely. If any part of the transaction fails, the entire transaction is rolled back preserving the original state of the directory.

consistency
: The transaction preserves the internal consistency of the database.

isolation
: The transaction is serialized by a global lock so that it is run independently of any other transactions.

durability
: The results of a committed transaction are backed up in stable storage, usually a disk.

Usage

Transactions are limited to a single connection to a single IBM Tivoli Directory server and are supported by the LDAP extended operations APIs.

Only one transaction at a time can be running over the same connection. During the transaction, no non-transactional operations can be issued over the same connection.

A transaction consists of three parts:

- An extended request to start the transaction
- Update operations:
  - add
  - modify
  - modify rdn
  - delete

  Note: The current release does not support some operations. For example, bind, unbind, search, extended `op`, and other operations. Referral objects can be updated only with `manageDsaIT` control specified.
- An extended request to end the transaction

To start a transaction, the client must send an extended request in the form of:

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {


requestValue [1] OCTET STRING OPTIONAL }
```

When the server receives the request, it generates a unique transaction ID. It then sends back an extended response in the form of:

```
ExtendedResponse ::= [APPLICATION 24]SEQUENCE{

COMPONENTS OF LDAPResult,

responseName [10] LDAPOID OPTIONAL,

response [11] OCTET STRING OPTIONAL }
```

The client submits subsequent update operations asynchronously with a control attached to all operations. The control contains the transaction ID returned in the `StartTransaction` response. The control has the form of:

```
Control ::= SEQUENCE {

controlType LDAPOID,

criticality BOOLEAN DEFAULT FALSE,

controlValue OCTET STRING OPTIONAL }
```

The server does not process update operations immediately. Instead, it saves the necessary information of operations in a queue.

The client sends an extended request to end the transaction that either commits or rolls back the transaction. If the server receives the commit operation result, it uses a global writer lock to serialize the transaction. It then retrieves the set of update operations that are identified by the transaction ID from the queue and begins to run these operations. If all operations succeed, the results are committed to the database and the server sends back the success return code.

As each operation is run, it generates a success return code unless an error occurs during the transaction. In this case, an unsuccessful return code is returned for all the operations. If any operation fails, the server rolls back the transaction and sends back the error return code of the failed operation. It sends to the operation in the client that caused the failure. The `EndTransaction` operation also receives an unsuccessful return code if the transaction is not successful. For any subsequent update operations that remain in the queue, an unsuccessful return code is generated. When the transaction times out, the connection is dropped and any subsequent operations receive an unsuccessful return code.

The server releases the global lock after the commit or the rollback is run. The event notification and change log operations are run only if the transaction succeeds.

Example

The `ldapmod.c` example file provides an understanding about the transaction capability.

The following example is an `ldapmod.c` example file, which is modified for limited transaction capability:

```
static char sccsid[] = "@(#)17  1.35 11/18/02 progref.idd, ldap, 5.2 15:20:20";
/*
 * COMPONENT_NAME: ldap.clients
 *
 * ABSTRACT: generic program to modify or add entries using LDAP with a transaction
 *
 * ORIGINS: 202,27
 *
 * (C) COPYRIGHT International Business Machines Corp. 2002
 * All Rights Reserved
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 */

/*
 * Copyright (c) 1995 Regents of the University of Michigan.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of Michigan at Ann Arbor. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided ``as is'' without express or implied warranty.
 */

/* ldaptxmod.c - generic program to modify or add entries using LDAP
using a single transaction */

#include <ldap.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>

#if !defined( WIN32 )
#include <sys/file.h>
#include <fcntl.h>
#include <unistd.h>
#endif
#define LDAPMODIFY_REPLACE 1
#define LDAPMODIFY_ADD  2

#if defined( WIN32 )
#define strcasecmp stricmp
#endif

#define safe_realloc( ptr, size ) ( ptr == NULL ? malloc( size ) : \
      realloc( ptr, size ))

#define MAX_SUPPLIED_PW_LENGTH 256
#define LDAPMOD_MAXLINE  4096

/* Strings found in replog/LDIF entries (mostly lifted from slurpd/slurp.h) */
#define T_REPLICA_STR  "replica"
#define T_DN_STR   "dn"
#define T_CHANGETYPESTR  "changetype"
#define T_ADDCTSTR   "add"
#define T_MODIFYCTSTR  "modify"
#define T_DELETECTSTR  "delete"
#define T_MODRDNCTSTR  "modrdn"
#define T_MODDNCTSTR   "moddn"
#define T_MODOPADDSTR  "add"
#define T_OPERSTR "transaction_operation"
#define T_MODOPREPLACESTR "replace"
#define T_MODOPDELETESTR "delete"
#define T_MODSEPSTR   "-"
#define T_NEWRDNSTR   "newrdn"
#define T_DELETEOLDRDNSTR "deleteoldrdn"
#define T_NEWSUPERIORSTR  "newsuperior"
#define T_CONTROLSTR "control"
```

```
extern char * str_getline(char**);
char * getPassword(void);
char * read_one_record(FILE *fp);

#if defined _WIN32
int getopt (int, char**, char*);
#endif
#ifndef -win32
#ifdef -GCC3
#include <errno.h>
#else
extern int errno;
#endif
#endif

/*Required for password prompting*/
#ifdef -win32
#include <conio.h>
#else
/*termios.h is defined by POSIX*/
#include <termios.h>
#endif

/* Global variables */
static LDAP     *ld         = NULL;  /* LDAP sesssion handle */
static FILE     *fp         = NULL;  /* input file handle */
static char *prog       = NULL;  /* program name */
static char *binddn     = NULL;  /* bind DN */
static char *passwd     = NULL;  /* bind password */
static char *ldaphost   = "localhost";  /* server host name */
static char     *mech       = NULL;  /* bind mechanism */
static char     *charset    = NULL;  /* character set for input */
static char     *keyfile    = NULL;  /* SSL key database file name*/
static char     *keyfile_pw = NULL;  /* SSL key database password */
static char     *cert_label = NULL;  /* client certificate label */
static int      hoplimit  = 10;      /* limit for referral chasing */
static int ldapport  = LDAP_PORT; /* server port number */
static int doit      = 1;       /* 0 to make believe */
static int verbose   = 0;       /* 1 for more trace messages */
static int contoper  = 0;       /* 1 to continue after errors */
static int force     = 0;
static int valsfromfiles = 0;
static int operation   = LDAPMODIFY_REPLACE;
static int referrals   = LDAP_OPT_ON;
static int ldapversion = LDAP_VERSION3;
static int DebugLevel  = 0;          /* 1 to activate library traces */
static int ssl         = 0;          /* 1 to use SSL */
static int  manageDsa  = LDAP_FALSE; /* LDAP_TRUE to modify referral objects */

static LDAPControl manageDsaIT = {
  "2.16.840.1.113730.3.4.2", /* OID */
  { 0, NULL },               /* no value */
  LDAP_OPT_ON                /* critical */
};

/* NULL terminated array of server controls*/
static LDAPControl *Server_Controls[3] = {NULL, NULL, NULL};

static int Num_Operations = 0;  /* count of times one must go to
        ldap_result to check result codes */
static int Message_ID = 0;      /* message ID returned by async
        ldap operation, currently not tracked*/
static int abort_flag = 0;      /* abort transaction flag set by
        -A parameter */

/* Implement getopt() for Windows to parse command line arguments. */
```

```
#if defined(_WIN32)
char *optarg = NULL;
int   optind = 1;
int   optopt = 0;
#define EMSG ""

int getopt(int argc, char **argv, char *ostr) {
  static char *place = EMSG;
  register char *oli;

  if (!*place) {
    if (optind >= argc || *(place = argv[optind]) != '-' || !*++place) {
      return EOF;
    }
    if (*place == '-') {
      ++optind;
      return EOF;
    }
  }
  if ((optopt = (int)*place++) == (int)':' || !(oli = strchr(ostr, optopt))) {
    if (!*place) {
      ++optind;
    }
    fprintf(stderr, "%s: %s: %c\n", "getopt", "illegal option", optopt);
    return ( '?' );
  }
  if (*++oli != ':') {
    optarg = NULL;
    if (!*place)
      ++optind;
  } else {
    if (*place) {
      optarg = place;
    } else if (argc <= ++optind) {
      place = EMSG;
      fprintf(stderr, "%s: %s: %c\n", "getopt", "option requires an argument",
 optopt);
      return 0;
    } else {
      optarg = argv[optind];
    }
    place = EMSG;
    ++optind;
  }
  return optopt;
}
#endif

/* Display usage statement and exit. */
void usage()
{
  fprintf(stderr, "\nSends modify or add requests to an LDAP server.\n");
  fprintf(stderr, "usage:\n");
  fprintf(stderr, "    %s [options] [-f file]\n", prog);
  fprintf(stderr, "where:\n");
  fprintf(stderr, "    file: name of input file\n");
  fprintf(stderr, "note:\n");
  fprintf(stderr, "    standard input is used if file is not specified\n");
  fprintf(stderr, "options:\n" );
  fprintf(stderr, "    -h host      LDAP server host name\n");
  fprintf(stderr, "    -p port      LDAP server port number\n");
  fprintf(stderr, "    -D dn        bind DN\n");
  fprintf(stderr, "    -w password  bind password or '?' for non-echoed prompt\n");
  fprintf(stderr, "    -Z           use a secure ldap connection (SSL)\n");
  fprintf(stderr, "    -K keyfile   file to use for keys\n");
  fprintf(stderr, "    -P key_pw    keyfile password\n");
  fprintf(stderr, "    -N key_name  private key name to use in keyfile\n");
  fprintf(stderr, "    -R           do not chase referrals\n");
  fprintf(stderr, "    -M           Manage referral objects as normal entries.\n");
  fprintf(stderr, "    -m mechanism perform SASL bind with the given mechanism\n");
  fprintf(stderr, "    -O maxhops   maximum number of referrals to follow in a
   sequence\n");
  fprintf(stderr, "    -V version   LDAP protocol version (2 or 3; only 3 is
   supported)\n");
  fprintf(stderr, "    -C charset   character set name to use, as registered with
   IANA\n");
  fprintf(stderr, "    -a           force add operation as default\n");
  fprintf(stderr, "    -r           force replace operation as default\n");
  fprintf(stderr, "    -b           support binary values from files (old style
   paths)\n");
  fprintf(stderr, "    -c           continuous operation; do not stop processing
   on error\n");
  fprintf(stderr, "    -n           show what would be done but don't actually do
   it\n");
  fprintf(stderr, "    -v           verbose mode\n");
  fprintf(stderr, "    -A           set transaction abort flag\n");
  fprintf(stderr, "    -d level     set debug level in LDAP library\n");
  exit(1);
}

/* Parse command line arguments. */
void parse_arguments(int argc, char **argv) {
  int i = 0;
  int port = 0;
  char *optpattern = "FaAbcRMZnrv?h:V:p:D:w:d:f:K:P:N:C:O:m:";
#ifndef _WIN32
  extern char *optarg;
  extern int optind;
#endif

  fp = stdin;
  while ((i = getopt(argc, argv, optpattern)) != EOF) {
    switch ( i ) {
    case 'V':
      ldapversion = atoi(optarg);
      if (ldapversion != LDAP_VERSION3) {
 fprintf(stderr, "Unsupported version level supplied.\n");
 usage();
      }
      break;
    case 'A':       /* force all changes records to be used */
      abort_flag = 1;
      break;
    case 'a':
      operation = LDAPMODIFY_ADD;
      break;
    case 'b': /* read values from files (for binary attributes)*/
      valsfromfiles = 1;
      break;
    case 'c': /* continuous operation*/
      contoper = 1;
      break;
    case 'F': /* force all changes records to be used*/
      force = 1;
      break;
    case 'h': /* ldap host*/
      ldaphost = strdup( optarg );
      break;
    case 'D': /* bind DN */
      binddn = strdup( optarg );
      break;
    case 'w': /* password*/
      if (optarg && optarg[0] == '?') {
 passwd = getPassword();
```

```
     } else
if (!(passwd = strdup( optarg )))
  perror("password");
     break;
   case 'd':
     DebugLevel = atoi(optarg);
     break;
   case 'f': /* read from file */
     if ((optarg[0] == '-') && (optarg[1] == '\0'))
fp = stdin;
     else if ((fp = fopen( optarg, "r" )) == NULL) {
perror( optarg );
exit( 1 );
     }
     break;
   case 'p':
     ldapport = atoi( optarg );
     port = 1;
     break;
   case 'n': /* print adds, don't actually do them*/
     doit = 0;
     break;
   case 'r': /* default is to replace rather than add values*/
     operation = LDAPMODIFY_REPLACE;
     break;
   case 'R':  /* don't automatically chase referrals*/
     referrals = LDAP_OPT_OFF;
     break;
   case 'M':   /* manage referral objects as normal entries */
     manageDsa = LDAP_TRUE;
     break;
   case 'O':   /* set maximum referral hop count  */
     hoplimit = atoi( optarg );
     break;
   case 'm':   /* use SASL bind mechanism  */
     if (!(mech = strdup ( optarg )))
perror("mech");
     break;
   case 'v':   /* verbose mode */
     verbose++;
     break;
   case 'K':
     keyfile = strdup( optarg );
     break;
   case 'P':
     keyfile_pw = strdup( optarg );
     break;
   case 'N':
     cert_label = strdup( optarg );
     break;
   case 'Z':
     ssl = 1;
     break;
   case 'C':
     charset = strdup(optarg);
     break;
   case '?':
   default:
     usage();
   }
 }

 if (argc - optind !=  0)
   usage();

 /* Use default SSL port if none specified*/
 if (( port == 0 ) && ( ssl ))
```

```
    ldapport = LDAPS_PORT;

  if ( ! DebugLevel ) {
    char *debug_ptr = NULL;

    if ( ( debug_ptr = getenv ( "LDAP_DEBUG" ) ) )
      DebugLevel = atoi ( debug_ptr );
  }
}

/* Get a password from the user but don't display it. */
char* getPassword( void ) {
  char supplied_password[ MAX_SUPPLIED_PW_LENGTH + 1 ]; /* Buffer for password */

#ifdef _WIN32
  char in  = '\0';                  /* Input character */
  int  len = 0;                     /* Length of password */
#else
  struct termios echo_control;
  struct termios save_control;

  int fd = 0;                  /* File descriptor */
  int attrSet = 0;             /* Checked later for reset */

  /* Get the file descriptor associated with stdin. */
  fd = fileno( stdin );

  if (tcgetattr( fd, &echo_control ) != -1) {
    save_control = echo_control;
    echo_control.c_lflag &= ~( ECHO | ECHONL );

    if (tcsetattr( fd, TCSANOW, &echo_control ) == -1) {
      fprintf(stderr, "Internal error setting terminal attribute.\n");
      exit( errno );
    }

    attrSet = 1;
  }
#endif

  /* Prompt for a password. */
  fputs( "Enter password ==> ", stdout );
  fflush( stdout );

#ifdef _WIN32
  /* Windows 9x/NT will always read from the console, i.e.,
     piped or redirected input will be ignored. */
  while ( in != '\r' && len <= MAX_SUPPLIED_PW_LENGTH ) {
    in = _getch();

    if (in != '\r') {
      supplied_password[len] = in;
      len++;
    } else {
      supplied_password[len] = '\0';
    }
  }
#else
  /* Get the password from stdin. */
  fgets( supplied_password, MAX_SUPPLIED_PW_LENGTH, stdin );

  /* Remove the newline at the end. */
  supplied_password[strlen( supplied_password ) - 1] = '\0';

#endif

#ifndef _WIN32
```

```
  /* Reset the terminal. */
  if (attrSet && tcsetattr( fd, TCSANOW, &save_control ) == -1) {
    fprintf(stderr, "Unable to reset the display.\n");
  }
#endif
  fprintf( stdout, "\n" );

  return ( supplied_password == NULL )? supplied_password :
   strdup( supplied_password );
}

/* Rebind callback function. */
int rebindproc(LDAP *ld, char **dnp, char **pwp, int *methodp, int freeit) {
  if ( !freeit ) {
    *methodp = LDAP_AUTH_SIMPLE;
    if ( binddn != NULL ) {
      *dnp = strdup( binddn );
      *pwp = strdup ( passwd );
    } else {
      *dnp = NULL;
      *pwp = NULL;
    }
  } else {
    free ( *dnp );
    free ( *pwp );
  }
  return LDAP_SUCCESS;
}

/* Connect and bind to server. */
void connect_to_server() {
  int failureReasonCode, rc, authmethod;
  struct berval  ber;
  struct berval  *server_creds;

  /* call ldap_ssl_client_init if V3 and SSL */
  if (ssl && (ldapversion == LDAP_VERSION3)) {
    if ( keyfile == NULL ) {
      keyfile = getenv("SSL_KEYRING");
      if (keyfile != NULL) {
 keyfile = strdup(keyfile);
      }
    }

    if (verbose)
      printf( "ldap_ssl_client_init( %s, %s, 0, &failureReasonCode )\n",
       ((keyfile) ? keyfile : "NULL"),
       ((keyfile_pw) ? keyfile_pw : "NULL"));
#ifdef LDAP_SSL_MAX
    rc = ibm_set_unrestricted_cipher_support();
    if (rc != 0) {
      fprintf( stderr, "Warning: ibm_gsk_set_unrestricted_cipher_support failed!
        rc == %d\n", rc );
    }
#endif

    rc = ldap_ssl_client_init( keyfile, keyfile_pw, 0, &failureReasonCode );
    if (rc != LDAP_SUCCESS) {
      fprintf( stderr,
        "ldap_ssl_client_init failed! rc == %d, failureReasonCode == %d\n",
        rc, failureReasonCode );
      exit( 1 );
    }
  }

  /* Open connection to server */
  if (ldapversion == LDAP_VERSION3) {
    if (ssl) {
      if (verbose)
 printf("ldap_ssl_init( %s, %d, %s )\n", ldaphost, ldapport,
        ((cert_label) ? cert_label : "NULL"));
      ld = ldap_ssl_init( ldaphost, ldapport, cert_label );
      if (ld == NULL) {
 fprintf( stderr, "ldap_ssl_init failed\n" );
 perror( ldaphost );
 exit( 1 );
      }
    } else {
      if (verbose)
 printf("ldap_init(%s, %d) \n", ldaphost, ldapport);
      if ((ld = ldap_init(ldaphost, ldapport)) == NULL) {
 perror(ldaphost);
 exit(1);
      }
    }
  }

  /* Set options */
  ldap_set_option (ld, LDAP_OPT_PROTOCOL_VERSION, (void * )&ldapversion);

  if (ldapversion == LDAP_VERSION3) {
    ldap_set_option (ld, LDAP_OPT_DEBUG, (void * )&DebugLevel);
    ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, (void *)&hoplimit);
  }
  ldap_set_option (ld, LDAP_OPT_REFERRALS, (void * )referrals);
  if (binddn != NULL)
    ldap_set_rebind_proc( ld, (LDAPRebindProc)rebindproc );
  if (charset != NULL) {
    if (ldap_set_iconv_local_charset(charset) != LDAP_SUCCESS) {
      fprintf(stderr, "unsupported charset %s\n", charset);
      exit(0);
    }
    ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_ON);
  }

  /* Bind to server */
  if (ldapversion == LDAP_VERSION3) {
    if ( ! mech ) /* Use simple bind */ {
      rc = ldap_simple_bind_s(ld, binddn, passwd);
      if ( rc != LDAP_SUCCESS ) {
 ldap_perror( ld, "ldap_simple_bind" );
 /* LDAP_OPT_EXT_ERROR only valuable for ssl communication.
    In this example, for LDAP v3, the bind is the first
    instance in which communication actually flows to the
    server.  So, if there is an ssl configuration error or
    other ssl problem, this will be the first instance where
    it will be detected. */
 if (ssl) {
   ldap_get_option( ld, LDAP_OPT_EXT_ERROR, &failureReasonCode);
   fprintf( stderr, "Attempted communication over SSL.\n");
   fprintf( stderr, "  The extended error is %d.\n", failureReasonCode);
 }
 exit( rc );
      }
    } else /* Presence of mechanism means SASL bind */ {
      /* Special case for mech="EXTERNAL".  Unconditionally set bind DN
  and credentials to NULL.  This option should be used in tandem
  with SSL and client authentication.  For other SASL mechanisms,
  use the specified bind DN and credentials. */
      if (strcmp(mech, LDAP_MECHANISM_EXTERNAL) == 0) {
 rc = ldap_sasl_bind_s (ld, NULL, mech, NULL, NULL, NULL, &server_creds);
 if (rc != LDAP_SUCCESS ) {
   ldap_perror ( ld, "ldap_sasl_bind_s" );
   exit( rc );
```

```
 }
      } else {
 if (strcmp(mech, LDAP_MECHANISM_GSSAPI) == 0) {
   rc = ldap_sasl_bind_s (ld, NULL, mech, NULL, NULL, NULL, &server_creds);
   if (rc != LDAP_SUCCESS ) {
     ldap_perror ( ld, "ldap_sasl_bind_s" );
     exit( rc );
   }
 } else /* other SASL mechanisms */ {
   ber.bv_len = strlen ( passwd );
   ber.bv_val = passwd;
   rc = ldap_sasl_bind_s (ld, binddn, mech, &ber, NULL, NULL, &server_creds);
   if (rc != LDAP_SUCCESS ) {
     ldap_perror ( ld, "ldap_sasl_bind_s" );
     exit( rc );
   }
 }
      }
    }
  }
}

/* Read a record from the file. */
char * read_one_record(FILE *fp)
{
  int len = 0;
  int lcur = 0;
  int   lmax = 0;
  char line[LDAPMOD_MAXLINE];
  char temp[LDAPMOD_MAXLINE];
  char *buf = NULL;

  /* Reads in and changes to ldif form */
  while (( fgets( line, sizeof(line), fp ) != NULL )) {
    if (!(strncmp(line,"changenumber",10)))
      {do
 fgets(line,sizeof(line),fp);
      while(strncmp(line,"targetdn",8)); /*changes the = to : for parse*/
      line[8]=':';}

    if (!(strncmp(line,"changetype",9)))
      line[10]=':';
    if (!(strncmp(line,"changetype:delete",16)))
      (fgets(temp,sizeof(line),fp)); /*gets rid of the changetime line after
   a delete.*/
    if (!(strncmp(line,"changetime",9)))
      {fgets(line,sizeof(line),fp);
      if (!(strncmp(line,"newrdn",6)))
 line[6]=':';
      else
 line[7]=':';
      }
    if (!(strncmp(line,"deleteoldrdn",12)))
      line[12]=':';
    if ( *line != '\n' ) {
      len = strlen( line );
      if ( lcur + len + 1 > lmax ) {
 lmax = LDAPMOD_MAXLINE
   *(( lcur + len + 1 ) / LDAPMOD_MAXLINE + 1 );
 if (( buf = (char *)safe_realloc( buf, lmax )) == NULL ) {
   perror( "safe_realloc" );
   exit( 1 );
 }
      }
      strcpy( buf + lcur, line );
      lcur += len;
    }
    else {
      if ( buf == NULL )
 continue; /* 1st line keep going */
      else
 break;
    }
  }

  return buf;
}

/* Read binary data from a file. */
int fromfile(char *path, struct berval *bv) {
  FILE *fp  = NULL;
  long  rlen = 0;
  int  eof  = 0;

  /* "r" changed to "rb", defect 39803. */
  if (( fp = fopen( path, "rb" )) == NULL ) {
    perror( path );
    return -1;
  }

  if ( fseek( fp, 0L, SEEK_END ) != 0 ) {
    perror( path );
    fclose( fp );
    return -1;
  }

  bv->bv_len = ftell( fp );

  if (( bv->bv_val = (char *)malloc( bv->bv_len )) == NULL ) {
    perror( "malloc" );
    fclose( fp );
    return -1;
  }

  if ( fseek( fp, 0L, SEEK_SET ) != 0 ) {
    perror( path );
    fclose( fp );
    return -1;
  }

  rlen = fread( bv->bv_val, 1, bv->bv_len, fp );
  eof = feof( fp );
  fclose( fp );

  if ( rlen != (bv->bv_len) ) {
    perror( path );
    return -1;
  }

  return bv->bv_len;
}

/* Read binary data from a file specified with a URL. */
int fromfile_url(char *value, struct berval *bv) {
  char *file = NULL;
  char *src  = NULL;
  char *dst  = NULL;

  if (strncmp(value, "file:///", 8))
    return -1;

  /* unescape characters */
  for (dst = src = &value[8]; (*src != '\0'); ++dst) {
    *dst = *src;
    if (*src++ != '%')
      continue;
    if ((*src >= '0') && (*src <= '9'))
      *dst = (*src++ - '0') << 4;
    else if ((*src >= 'a') && (*src <= 'f'))
      *dst = (*src++ - 'a' + 10) << 4;
    else if ((*src >= 'A') && (*src <= 'F'))
      *dst = (*src++ - 'A' + 10) << 4;
    else
      return -1;
    if ((*src >= '0') && (*src <= '9'))
      *dst += (*src++ - '0');
    else if ((*src >= 'a') && (*src <= 'f'))
      *dst += (*src++ - 'a' + 10);
    else if ((*src >= 'A') && (*src <= 'F'))
      *dst += (*src++ - 'A'+ 10);
    else
      return -1;
  }
  *dst = '\0';

  /* On WIN32 platforms the URL must begin with a drive letter.
     On UNIX platforms the initial '/' is kept to indicate absolute
     file path.
  */
#ifdef _WIN32
  file = value + 8;
#else
  file = value + 7;
#endif
  return fromfile(file, bv);
}

/* Add operation to the modify structure. */
void addmodifyop(LDAPMod ***pmodsp, int modop, char *attr,
   char *value, int vlen, int isURL, int isBase64)
{
  LDAPMod **pmods = NULL;
  int i = 0;
  int j = 0;
  struct berval *bvp = NULL;

  /* Data can be treated as binary (wire ready) if one of the
     following applies:
     1) it was base64 encoded
     2) charset is not defined
     3) read from an external file
  */
  if (isBase64 ||
      (charset == NULL) ||
      isURL ||
      ((value != NULL) && valsfromfiles && (*value == '/'))) {
    modop |= LDAP_MOD_BVALUES;
  }

  i = 0;
  pmods = *pmodsp;
  if ( pmods != NULL ) {
    for (; pmods[ i ] != NULL; ++i ) {
      if ( strcasecmp( pmods[ i ]->mod_type, attr ) == 0 &&
    pmods[ i ]->mod_op == modop ) {
 break;
      }
    }
  }

  if ( pmods == NULL || pmods[ i ] == NULL ) {
    if (( pmods = (LDAPMod * *)safe_realloc( pmods, (i + 2) *
          sizeof( LDAPMod * ))) == NULL ) {
      perror( "safe_realloc" );
      exit( 1 );
    }
    *pmodsp = pmods;
    pmods[ i + 1 ] = NULL;
    if (( pmods[ i ] = (LDAPMod * )calloc( 1, sizeof( LDAPMod ))) == NULL ) {
      perror( "calloc" );
      exit( 1 );
    }
    pmods[ i ]->mod_op = modop;
    if (( pmods[ i ]->mod_type = strdup( attr )) == NULL ) {
      perror( "strdup" );
      exit( 1 );
    }
  }

  if ( value != NULL ) {
    if (modop & LDAP_MOD_BVALUES) {
      j = 0;
      if ( pmods[ i ]->mod_bvalues != NULL ) {
 for (; pmods[ i ]->mod_bvalues[ j ] != NULL; ++j ) {
   ;
 }
      }
      if (( pmods[ i ]->mod_bvalues =
     (struct berval **)safe_realloc( pmods[ i ]->mod_bvalues,
         (j + 2) * sizeof( struct berval *))) == NULL ) {
 perror( "safe_realloc" );
 exit( 1 );
      }

      pmods[ i ]->mod_bvalues[ j + 1 ] = NULL;
      if (( bvp = (struct berval *)malloc( sizeof( struct berval )))
   == NULL ) {
 perror( "malloc" );
 exit( 1 );
      }
      pmods[ i ]->mod_bvalues[ j ] = bvp;

      /* get value from file */
      if ( valsfromfiles && *value == '/' ) {
 if (fromfile( value, bvp ) < 0 )
   exit(1);
      } else if (isURL) {
 if (fromfile_url(value, bvp) < 0)
   exit(1);
      } else {
 bvp->bv_len = vlen;
 if (( bvp->bv_val = (char *)malloc( vlen + 1 )) == NULL ) {
   perror( "malloc" );
   exit( 1 );
 }
 memmove( bvp->bv_val, value, vlen );
 bvp->bv_val[ vlen ] = '\0';
      }
    } else {
      j = 0;
      if ( pmods[ i ]->mod_values != NULL ) {
 for ( ; pmods[ i ]->mod_values[ j ] != NULL; ++j ) {
   ;
 }
      }
      if (( pmods[ i ]->mod_values =
     (char **)safe_realloc( pmods[ i ]->mod_values,
       (j + 2) * sizeof( char *))) == NULL ) {
```

```
 perror( "safe_realloc" );
 exit( 1 );
      }
      pmods[ i ]->mod_values[ j + 1 ] = NULL;
      if (( pmods[ i ]->mod_values[ j ] = strdup( value )) == NULL) {
 perror( "strdup" );
 exit( 1 );
      }
    }
  }
}

/* Delete record */
int dodelete( char *dn ) {
  int rc = 0;

  printf( "%sdeleting entry %s\n", (!doit) ? "!" : "", dn );
  if (!doit)
    return LDAP_SUCCESS;

  rc = ldap_delete_ext( ld, dn,
   Server_Controls,
   NULL, &Message_ID);
  if ( rc != LDAP_SUCCESS )
    ldap_perror( ld, "ldap_delete" );
  else
    printf( "delete complete\n" );

  putchar('\n');
  /* Increment results to check after end transaction. */
  Num_Operations++;
  return rc;
}

/* Copy or move an entry. */
int domodrdn( char *dn, char *newrdn, int deleteoldrdn ) {
  int rc = 0;

  printf( "%s%s %s to %s\n", ((!doit) ? "!" : ""),
   ((deleteoldrdn) ? "moving" : "copying"), dn, newrdn);
  if (!doit)
    return LDAP_SUCCESS;

  rc = ldap_rename( ld, dn, newrdn, NULL, deleteoldrdn,
      Server_Controls , NULL,
      &Message_ID );
  if ( rc != LDAP_SUCCESS )
    ldap_perror( ld, "ldap_rename" );
  else
    printf( "rename operation complete\n" );
  putchar('\n');

  /* Increment the count of results to check after end transaction is sent */
  Num_Operations++;
  return rc;
}

/* Print a binary value. If charset is not specified then check to
   see if string is printable anyway. */
void print_binary(struct berval *bval) {
  int i = 0;
  int binary = 0;

  printf( "\tBINARY (%ld bytes) ", bval->bv_len);
  if (charset == NULL) {
    binary = 0;
    for (i = 0; (i < (bval->bv_len)) && (!binary); ++i)
```

```
      if (!isprint(bval->bv_val[i]))
 binary = 1;
    if (!binary)
      for (i = 0; (i < (bval->bv_len)); ++i)
 putchar(bval->bv_val[i]);
  }
  putchar('\n');
}

/* Modify or add an entry. */
int domodify( char *dn, LDAPMod **pmods, int newentry ) {
  int i, j, op, rc;
  struct berval *bvp;

  if ( pmods == NULL ) {
    fprintf( stderr, "%s: no attributes to change or add (entry %s)\n",
      prog, dn );
    return LDAP_PARAM_ERROR;
  }

  if ( verbose ) {
    for ( i = 0; pmods[ i ] != NULL; ++i ) {
      op = pmods[ i ]->mod_op & ~LDAP_MOD_BVALUES;
      printf( "%s %s:\n", op == LDAP_MOD_REPLACE ?
       "replace" : op == LDAP_MOD_ADD ?
       "add" : "delete", pmods[ i ]->mod_type );
      if (pmods[i]->mod_op & LDAP_MOD_BVALUES) {
 if (pmods[ i ]->mod_bvalues != NULL) {
   for (j = 0; pmods[i]->mod_bvalues[j] != NULL; ++j)
     print_binary(pmods[i]->mod_bvalues[j]);
 }
      } else {
 if (pmods[i]->mod_values != NULL) {
   for (j = 0; pmods[i]->mod_values[j] != NULL; ++j)
     printf("\t%s\n", pmods[i]->mod_values[j]);
 }
      }
    }
  }

  if ( newentry )
    printf( "%sadding new entry %s as a transaction\n", (!doit) ? "!" : "", dn );
  else
    printf( "%smodifying entry %s as a transaction\n", (!doit) ? "!" : "", dn );
  if (!doit)
    return LDAP_SUCCESS;

  if ( newentry ) {
    rc = ldap_add_ext( ld, dn, pmods,
         Server_Controls, NULL,
         &Message_ID);
  } else {
    rc = ldap_modify_ext( ld, dn, pmods,
     Server_Controls, NULL,
     &Message_ID );
  }
  if ( rc != LDAP_SUCCESS ) {
    ldap_perror( ld, newentry ? "ldap_add" : "ldap_modify" );
  } else if ( verbose ) {
    printf( "%s operation complete\n", newentry ? "add" : "modify" );
  }
  putchar( '\n' );

  /* Increment the count of results to check after end transaction is sent */
  Num_Operations++;
  return rc;
}
```

```
/* Process an ldif record. */
int process_ldif_rec(char *rbuf) {
  char *line     = NULL;
  char *dn       = NULL;
  char *type     = NULL;
  char *value    = NULL;
  char *newrdn   = NULL;
  char *p        = NULL;
  int is_url   = 0;
  int   is_b64   = 0;
  int rc       = 0;
  int   linenum  = 0;
  int   vlen     = 0;
  int   modop    = 0;
  int   replicaport   = 0;
  int expect_modop  = 0;
  int   expect_sep    = 0;
  int   expect_ct     = 0;
  int   expect_newrdn = 0;
  int expect_deleteoldrdn = 0;
  int   deleteoldrdn  = 1;
  int saw_replica   = 0;
  int use_record      = force;
  int new_entry = (operation == LDAPMODIFY_ADD);
  int delete_entry = 0;
  int got_all = 0;
  LDAPMod **pmods = NULL;
  int version = 0;
  int str_rc  = 0;

  while ( rc == 0 && ( line = str_getline( &rbuf )) != NULL ) {
    ++linenum;

    /* Is this a separator line ("-")? */
    if ( expect_sep && strcasecmp( line, T_MODSEPSTR ) == 0 ) {
      /* If modifier has not been added yet then go ahead and add
  it. The can happen on sequences where there are no
  attribute values, such as:
  DELETE: title
  -
      */
      if (value != NULL)
 addmodifyop(&pmods, modop, value, NULL, 0, 0, 0);
      value = NULL;
      expect_sep = 0;
      expect_modop = 1;
      continue;
    }

    str_rc = str_parse_line_v_or_bv(line, &type, &value, &vlen, 1, &is_url,
   &is_b64);
    if ((strncmp(type,"changes",7))==0)
      {str_parse_line_v_or_bv(value, &type, &value, &vlen, 1, &is_url, &is_b64);}
    if ((linenum == 1) && (strcmp(type, "version") == 0)) {
      version = atoi(value);
      continue;
    }

    if ((linenum == 2) && (version == 1) &&
 (strcmp(type, "charset") == 0)) {
      if (charset != NULL)
 free(charset);
      charset = strdup(value);
      if ((rc = ldap_set_iconv_local_charset(charset)) != LDAP_SUCCESS) {
 fprintf(stderr, "unsupported charset %s\n", charset);
 break;
```

```
      }
      ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_ON);
      continue;
    }

    if ( dn == NULL ) {
      if ( !use_record && strcasecmp( type, T_REPLICA_STR ) == 0 ) {
++saw_replica;
if (( p = strchr( value, ':' )) == NULL ) {
  replicaport = LDAP_PORT;
} else {
  *p++ = '\0';
  replicaport = atoi( p );
}
if ( strcasecmp( value, ldaphost ) == 0 &&
     replicaport == ldapport ) {
  use_record = 1;
}
      } else if ( strcasecmp( type, T_DN_STR ) == 0 ) {
if (( dn = strdup( value )) == NULL ) {
  perror( "strdup" );
  exit( 1 );
}
expect_ct = 1;
      }
      continue; /* skip all lines until we see "dn:" */
    }

    if ( expect_ct ) {
      expect_ct = 0;
      if ( !use_record && saw_replica ) {
printf( "%s: skipping change record for entry: %s\n\t(LDAP host/port does
       not match replica: lines)\n", prog, dn );
free( dn );
return 0;
      }

      /* this is an ldif-change-record */
      if ( strcasecmp( type, T_CHANGETYPESTR ) == 0 ) {
if ( strcasecmp( value, T_MODIFYCTSTR ) == 0 ) {
  new_entry = 0;
  expect_modop = 1;
} else if ( strcasecmp( value, T_ADDCTSTR ) == 0 ) {
  modop = LDAP_MOD_ADD;
  new_entry = 1;
} else if ( strcasecmp( value, T_MODRDNCTSTR ) == 0 ) {
  expect_newrdn = 1;
} else if ( strcasecmp( value, T_DELETECTSTR ) == 0 ) {
  got_all = delete_entry = 1;
} else {
  fprintf( stderr,
    "%s:  unknown %s \"%s\" (line %d of entry: %s)\n",
    prog, T_CHANGETYPESTR, value, linenum, dn );
  rc = LDAP_PARAM_ERROR;
}
continue;

/* this is an ldif-attrval-record */
      } else {
if (operation == LDAPMODIFY_ADD) {
  new_entry = 1;
  modop = LDAP_MOD_ADD;
} else
  modop = LDAP_MOD_REPLACE;
      }
    }
```

```
   if (expect_modop) {
     expect_modop = 0;
     expect_sep = 1;
     if ( strcasecmp( type, T_MODOPADDSTR ) == 0 ) {
modop = LDAP_MOD_ADD;
continue;
     } else if ( strcasecmp( type, T_MODOPREPLACESTR ) == 0 ) {
modop = LDAP_MOD_REPLACE;
continue;
     } else if ( strcasecmp( type, T_MODOPDELETESTR ) == 0 ) {
modop = LDAP_MOD_DELETE;
continue;
     } else {
fprintf(stderr,
 "%s: unknown mod_spec \"%s\" (line %d of entry: %s)\n",
 prog, type, linenum, dn);
rc = LDAP_PARAM_ERROR;
continue;
     }
   }

   if ( expect_newrdn ) {
     if ( strcasecmp( type, T_NEWRDNSTR ) == 0 ) {
if (( newrdn = strdup( value )) == NULL ) {
  perror( "strdup" );
  exit( 1 );
}
expect_deleteoldrdn = 1;
expect_newrdn = 0;
     } else {
fprintf( stderr, "%s: expecting \"%s:\" but saw \"%s:\" (line %d of entry %s)\n",
  prog, T_NEWRDNSTR, type, linenum, dn );
rc = LDAP_PARAM_ERROR;
     }
   } else if ( expect_deleteoldrdn ) {
     if ( strcasecmp( type, T_DELETEOLDRDNSTR ) == 0 ) {
deleteoldrdn = ( *value == '0' ) ? 0 : 1;
got_all = 1;
     } else {
fprintf( stderr, "%s: expecting \"%s:\" but saw \"%s:\" (line %d of entry %s)\n",
  prog, T_DELETEOLDRDNSTR, type, linenum, dn );
rc = LDAP_PARAM_ERROR;
     }
   } else if ( got_all ) {
     fprintf( stderr, "%s: extra lines at end (line %d of entry %s)\n",
       prog, linenum, dn );
     rc = LDAP_PARAM_ERROR;
   } else {

     addmodifyop(&pmods, modop, type, value, vlen, is_url, is_b64);
     type = NULL;
     value = NULL;
   }
 }

 /* If last separator is missing go ahead and handle it anyway, even
    though it is technically invalid ldif format. */
 if (expect_sep && (value != NULL))
   addmodifyop(&pmods, modop, value, NULL, 0, 0, 0);

 if ( rc == 0 ) {
   if (delete_entry)
     rc = dodelete( dn );

   else if (newrdn != NULL)
     rc = domodrdn( dn, newrdn, deleteoldrdn );
   else if (dn != NULL)
```

```
      rc = domodify( dn, pmods, new_entry );
  }

  if (dn != NULL)
    free( dn );
  if ( newrdn != NULL )
    free( newrdn );
  if ( pmods != NULL )
    ldap_mods_free( pmods, 1 );

  return rc;
}

/* Process a mod record. */
int process_ldapmod_rec( char *rbuf ) {
  char *line      = NULL;
  char *dn        = NULL;
  char *p         = NULL;
  char *q         = NULL;
  char *attr      = NULL;
  char *value     = NULL;
  int rc        = 0;
  int   linenum   = 0;
  int   modop     = 0;
  LDAPMod **pmods = NULL;

  while ( rc == 0 && rbuf != NULL && *rbuf != '\0' ) {
    ++linenum;
    if (( p = strchr( rbuf, '\n' )) == NULL ) {
      rbuf = NULL;
    } else {
      if ( *(p - 1) == '\\' ) { /* lines ending in '\' are continued */
 strcpy( p - 1, p );
 rbuf = p;
 continue;
      }
      *p++ = '\0';
      rbuf = p;
    }

    if ( dn == NULL ) { /* first line contains DN */
      if (( dn = strdup( line )) == NULL ) {
 perror( "strdup" );
 exit( 1 );
      }
    } else {
      if (( p = strchr( line, '=' )) == NULL ) {
 value = NULL;
 p = line + strlen( line );
      } else {
 *p++ = '\0';
 value = p;
      }

      for ( attr = line; *attr != '\0' && isspace( *attr ); ++attr ) {
 ; /* skip attribute leading white space */
      }

      for ( q = p - 1; q > attr && isspace( *q ); --q ) {
 *q = '\0'; /* remove attribute trailing white space */
      }

      if ( value != NULL ) {
 while ( isspace( *value )) {
   ++value;  /* skip value leading white space */
 }
 for ( q = value + strlen( value ) - 1; q > value &&
  isspace( *q ); --q ) {
   *q = '\0'; /* remove value trailing white space */
 }
 if ( *value == '\0' ) {
   value = NULL;
 }
      }

      if ((value == NULL) && (operation == LDAPMODIFY_ADD)) {
 fprintf( stderr, "%s: missing value on line %d (attr is %s)\n",
   prog, linenum, attr );
 rc = LDAP_PARAM_ERROR;
      } else {
 switch ( *attr ) {
 case '-':
   modop = LDAP_MOD_DELETE;
   ++attr;
   break;
 case '+':
   modop = LDAP_MOD_ADD;
   ++attr;
   break;
 default:
   modop = (operation == LDAPMODIFY_REPLACE)
     ? LDAP_MOD_REPLACE : LDAP_MOD_ADD;
   break;
 }

 addmodifyop( &pmods, modop, attr, value,
       ( value == NULL ) ? 0 : strlen( value ), 0, 0);
      }
    }
    line = rbuf;
  }

  if ( rc == 0 ) {
    if ( dn == NULL )
      rc = LDAP_PARAM_ERROR;
    else
      rc = domodify(dn, pmods, (operation == LDAPMODIFY_ADD));
  }

  if ( pmods != NULL )
    ldap_mods_free( pmods, 1 );
  if ( dn != NULL )
    free( dn );

  return rc;
}

main( int argc, char **argv ) {
  char *rbuf = NULL;
  char *start = NULL;
  char *p = NULL;
  char *q = NULL;
  char *tmpstr = NULL;
  int rc = 0;
  int   i = 0;
  int   use_ldif = 0;
  int   num_checked = 0;
  char  *Start_Transaction_OID      = LDAP_START_TRANSACTION_OID;
  char  *End_Transaction_OID        = LDAP_END_TRANSACTION_OID;
  char  *Control_Transaction_OID    = LDAP_TRANSACTION_CONTROL_OID;
  char  *Returned_OID               = NULL;
  struct berval *Returned_BerVal    = NULL;
  struct berval Request_BerVal      = {0,0};
  char  *Berval                     = NULL;
  LDAPMessage *LDAP_result          = NULL;

  /* Strip off any path info on program name */
#if defined( _WIN32 )
  if ((prog = strrchr(argv[0], '\\')) != NULL)
    ++prog;
  else
    prog = argv[0];
#else
  if (prog = strrchr(argv[0], '/'))
    ++prog;
  else
    prog = argv[0];
#endif

#if defined( _WIN32 )
  /* Convert string to lowercase */
  for (i = 0; prog[i] != '\0'; ++i)
    prog[i] = tolower(prog[i]);

  /* Strip ending .exe from program name */
  if ((tmpstr = strstr(prog, ".exe")) != NULL)
    *tmpstr = '\0';
#endif
  if ( strcmp( prog, "ldaptxadd" ) == 0 )
    operation = LDAPMODIFY_ADD;

  /* Parse command line arguments. */
  parse_arguments(argc, argv);

  /* Connect to server. */
  if (doit)
    connect_to_server();

  /* Disable translation if reading from file (they must specify the
     translation in the file). */
  if (fp != stdin)
    ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_OFF);

  /* Do the StartTransaction extended operation.
     The transaction ID returned must be put into the server control
     sent with all update operations. */
  rc = ldap_extended_operation_s ( ld, Start_Transaction_OID,
       &Request_BerVal, NULL, NULL,
       &Returned_OID,
       &Returned_BerVal);
  if (verbose) {
    printf("ldap_extended_operation(start transaction) RC=%d\n", rc);
  }

  if ( rc != LDAP_SUCCESS) {
    fprintf(stderr, "Start transaction rc=%d -> %s\n",
     rc, ldap_err2string(rc));
    exit( rc );
  }

  /* Allocate the server control for transactions. */
  if (( Server_Controls[0] =
 (LDAPControl *)malloc( sizeof( LDAPControl ))) == NULL ) {
    perror("malloc");
    exit( 1 );
  }

  /* Allocate the server control's berval. */
  if ((Server_Controls[0]->ldctl_value.bv_val =
       (char *) calloc (1, Returned_BerVal->bv_len + 1)) == NULL) {
    perror("calloc");
    exit(1);
  }

  /* Copy the returned berval length and value into the server control */
  Server_Controls[0]->ldctl_value.bv_len = Returned_BerVal-> bv_len;
  memcpy(Server_Controls[0]->ldctl_value.bv_val,
  Returned_BerVal->bv_val , Returned_BerVal->bv_len);

  /* Set the control type to Transaction_Control_OID */
  Server_Controls[0]->ldctl_oid = Control_Transaction_OID;

  /* Set the criticality in the control to TRUE */
  Server_Controls[0]->ldctl_iscritical = LDAP_OPT_ON;

  /* If referral objects are to be modified directly, */
  if (manageDsa == LDAP_TRUE) {
    /* then set that server control as well. */
    Server_Controls[1] = &manageDsaIT
  }

  /* Initialize the count of operations that will be in the transaction.
     This count will be incremented by each operation that is performed.
     The count will be the number of calls that must be made to ldap_result
     to get the results for the operations.
  */
  Num_Operations = 0;

  /* Do operations */
  rc = 0;
  while ((rc == 0 || contoper) && (rbuf = read_one_record( fp )) != NULL ) {
    /* We assume record is ldif/slapd.replog if the first line
       has a colon that appears to the left of any equal signs, OR
       if the first line consists entirely of digits (an entry ID). */

    use_ldif=1;
    start = rbuf;

    if ( use_ldif )
      rc = process_ldif_rec( start );
    else
      rc = process_ldapmod_rec( start );
    free( rbuf );
  }

  /* Finish the transaction, committing or rolling back based on input parameter. */
  rc = 0;
  Request_BerVal.bv_len = Returned_BerVal->bv_len + 1;
  if ((Berval =
       ( char *) malloc (Returned_BerVal->bv_len + 1)) == NULL) {
    perror("malloc");
    exit(1);
  }

  memcpy (&Berval[1], Returned_BerVal->bv_val, Returned_BerVal->bv_len);
  Berval[0] = abort_flag ? '\1' : '\0';
  Request_BerVal.bv_val = Berval;

  rc = ldap_extended_operation_s ( ld,
       End_Transaction_OID,
     &Request_BerVal, NULL, NULL,
       &Returned_OID,
       &Returned_BerVal);
  if (verbose) {
    printf("ldap_extended_operation(end transaction) RC=%d\n", rc);
  }

  if ( rc != LDAP_SUCCESS) {
    fprintf(stderr, "End transaction rc=%d -> %s\n",
     rc, ldap_err2string(rc));
    exit( rc );
  }

  /* Process the results of the operations in the transaction.
     At this time we will not be concerned about the correctness
     of the message numbers, just whether the operations succceeded or not.
     We could keep track of the operation types and make sure they are all
     accounted for. */

  for ( num_checked = 0; num_checked < Num_Operations; num_checked++ ) {
    if (verbose) {
      printf("processing %d of %d operation results\n",
      1 + num_checked, Num_Operations);
    }

    rc = ldap_result (ld , LDAP_RES_ANY, LDAP_MSG_ONE, NULL, &LDAP_result);
    if ( rc <= 0) {
      if (rc == 0)
 fprintf(stderr, "Operation %d timed out\n", num_checked);
      if (rc < 0 )
 fprintf(stderr, "Operation %d failed\n", num_checked);
      exit( 1 );
    }
  }

  /* Unbind and exit */
  if (doit)
    ldap_unbind(ld);

  exit(0);
}
```

The following example shows the makefile:

```
#-----------------------------------------------------------------------------
# COMPONENT_NAME: examples
#
# ABSTRACT: makefile to generate LDAP client programs for transactions
#
# ORIGINS: 202,27
#
# (C) COPYRIGHT International Business Machines Corp. 2002
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#############################################################################
# Default definitions
#############################################################################
CC = cl.exe
LD      = link.exe
RM = erase /f
HARDLN = copy
### Note: Your install path may be different
LDAPHOME = D:\Program Files\IBM\ldap\V6.3.1

#############################################################################
# General compiler options
#############################################################################

DEFINES = /DNDEBUG /DWIN32 /D_CONSOLE /D_MBCS /DNT /DNEEDPROTOS
INCLUDES= /I"$(LDAPHOME)/include"
CFLAGS = /nologo /MD /GX /Z7 $(INCLUDES) $(DEFINES)
```

```
##############################################################################
# General linker options
##############################################################################

LIBS    = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
 advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib\
 odbccp32.lib wsock32.lib

# Use the following definition to link the sample programs statically.
#CLIENT_LIBS = ldapstatic.lib libidsldifstatic.lib setloci.lib iconvi.lib

# Use the following definition to link the sample programs with
# the LDAP shared library.
CLIENT_LIBS = ldap.lib libldif.lib setloci.lib
LDIR = /LIBPATH:"$(LDAPHOME)"/lib
LFLAGS  = /nologo /subsystem:console /incremental:no  \
 $(LDIR) $(LIBS) $(CLIENT_LIBS)

##############################################################################
# Targets
##############################################################################

all: ldaptxmod.exe ldaptxadd.exe

ldaptxmod.exe: ldaptxmod.obj
 $(LD) $(LFLAGS) /out:$@ $**

ldaptxadd.exe: ldaptxmod.exe
 $(RM) $@
 $(HARDLN) ldaptxmod.exe ldaptxadd.exe

.c.obj::
   $(CC) $(CFLAGS) /c $<

ldaptxmod.obj: ldaptxmod.c

clean:
 $(RM) ldaptxmod.exe ldaptxadd.exe ldaptxmod.obj
```

See the source file, `ldapmodify.c`, in the *DS_INSTALL_ROOT*/`examples` for more information about transaction.

Index

Notices

Trademarks

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Printed in USA

SC27-2754-02