Network Working Group
INTERNET-DRAFT
Intended Category: Standards Track
Obsoletes: RFC 1823
Expires: January 1998

T. Howes
Netscape Communications Corp.
M. Smith
Netscape Communications Corp.
A. Herron
Microsoft Corp.
C. Weider
Microsoft Corp.
M. Wahl
Critical Angle, Inc.

29 July 1997

The C LDAP Application Program Interface
<draft-ietf-asid-ldap-c-api-00.txt>

1. Status of this Memo

This draft document will be submitted to the RFC Editor as a Standards Track document. Distribution of this memo is unlimited. Please send comments to the authors.

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

2. Introduction

This document defines a C language application program interface to the lightweight directory access protocol (LDAP). This document replaces the previous definition of this API, defined in RFC 1823, updating it to include support for features found in version 3 of the LDAP protocol. New extended operation functions were added to support LDAPv3 features such as controls. In addition, other LDAP API changes were made to

support information hiding and thread safety.

The C LDAP API is designed to be powerful, yet simple to use. It defines compatible synchronous and asynchronous interfaces to LDAP to suit a wide variety of applications. This document gives a brief overview of the LDAP model, then an overview of how the API is used by an application program to obtain LDAP information. The API calls are described in detail, followed by an appendix that provides some example code demonstrating the use of the API. This document provides information to the Internet community. It does not specify any standard.

3. Overview of the LDAP Model

LDAP is the lightweight directory access protocol, described in [2] and [6]. It can provide a lightweight frontend to the X.500 directory [1], or a stand-alone service. In either mode, LDAP is based on a client-server model in which a client makes a TCP connection to an LDAP server, over which it sends requests and receives responses.

The LDAP information model is based on the entry, which contains information about some object (e.g., a person). Entries are composed of attributes, which have a type and one or more values. Each attribute has a syntax that determines what kinds of values are allowed in the attribute (e.g., ASCII characters, a jpeg photograph, etc.) and how those values behave during directory operations (e.g., is case significant during comparisons).

Entries may be organized in a tree structure, usually based on political, geographical, and organizational boundaries. Each entry is uniquely named relative to its sibling entries by its relative distinguished name (RDN) consisting of one or more distinguished attribute values from the entry. At most one value from each attribute may be used in the RDN. For example, the entry for the person Babs Jensen might be named with the "Barbara Jensen" value from the commonName attribute.

A globally unique name for an entry, called a distinguished name or DN, is constructed by concatenating the sequence of RDNs from the entry up to the root of the tree. For example, if Babs worked for the University of Michigan, the DN of her U-M entry might be "cn=Barbara Jensen, o=University of Michigan, c=US". The DN format used by LDAP is defined in [4].

Operations are provided to authenticate, search for and retrieve information, modify information, and add and delete entries from the tree. The next sections give an overview of how the API is used and detailed descriptions of the LDAP API calls that implement all of these functions.

4. Overview of LDAP API Use

An application generally uses the C LDAP API in four simple steps.

- Initialize an LDAP session with a default LDAP server. The ldap_init() function returns a handle to the session, allowing multiple connections to be open at once.

- Authenticate to the LDAP server. The ldap_bind() function and friends support a variety of authentication methods.

- Perform some LDAP operations and obtain some results. ldap_search() and friends return results which can be parsed by ldap_result2error(), ldap_first_entry(), ldap_next_entry(), etc.

- Close the session. The ldap_unbind() function closes the connection.

Operations can be performed either synchronously or asynchronously. The names of the synchronous functions end in _s. For example, a synchronous search can be completed by calling ldap_search_s(). An asynchronous search can be initiated by calling ldap_search(). All synchronous routines return an indication of the outcome of the operation (e.g, the constant LDAP_SUCCESS or some other error code). The asynchronous routines return the message id of the operation initiated. This id can be used in subsequent calls to ldap_result() to obtain the result(s) of the operation. An asynchronous operation can be abandoned by calling ldap_abandon().

Results and errors are returned in an opaque structure called LDAPMessage. Routines are provided to parse this structure, step through entries and attributes returned, etc. Routines are also provided to interpret errors. Later sections of this document describe these routines in more detail.

LDAP version 3 servers may return referrals to other servers. By default, implementations of this API will attempt to follow referrals automatically for the application. This behavior can be disabled globally (using the ldap_set_option() call) or on a per-request basis through the use of a client control.

As in the LDAPv3 protocol itself, all DNs and string values that are passed into or produced by the C LDAP API are represented as UTF-8[10] characters.

For compatibility with existing applications, implementations of this API will by default use version 2 of the LDAP protocol. Applications that intend to take advantage of LDAP version 3 features will need to

use the ldap_set_option() call with a LDAP_OPT_PROTOCOL_VERSION to switch to version 3.

5. Common Data Structures

Some data structures that are common to several LDAP API functions are defined here:

```
typedef struct ldap LDAP;

typedef struct ldapmsg LDAPMessage;

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

struct timeval {
        long            tv_sec;
        long            tv_usec;
};
```

The LDAP structure is an opaque data type that represents an LDAP session Typically this corresponds to a connection to a single server, but it may encompass several server connections in the face of LDAPv3 referrals.

The LDAPMessage structure is an opaque data type that is used to return results and error information.

The berval structure is used to represent arbitrary binary data and its fields have the following meanings:

bv_len Length of data in bytes.

bv_val A pointer to the data itself.

The timeval structure is used to represent an interval of time and its fields have the following meanings:

tv_sec Seconds component of time interval.

tv_usec Microseconds component of time interval.

6. LDAP Error Codes

Many of the LDAP API routines return LDAP error codes, some of which indicate local errors and some of which may be returned by servers. Supported error codes are (hexadecimal values are given in parentheses after the constant):

```
            LDAP_SUCCESS (0x00)
            LDAP_OPERATIONS_ERROR( 0x01)
            LDAP_PROTOCOL_ERROR (0x02)
            LDAP_TIMELIMIT_EXCEEDED (0x03)
            LDAP_SIZELIMIT_EXCEEDED (0x04)
            LDAP_COMPARE_FALSE (0x05)
            LDAP_COMPARE_TRUE (0x06)
            LDAP_STRONG_AUTH_NOT_SUPPORTED (0x07)
            LDAP_STRONG_AUTH_REQUIRED (0x08)
            LDAP_REFERRAL (0x0a)                              -- new in LDAPv3
            LDAP_ADMINLIMIT_EXCEEDED (0x0b)                   -- new in LDAPv3
            LDAP_UNAVAILABLE_CRITICAL_EXTENSION (0x0c)        -- new in LDAPv3
            LDAP_CONFIDENTIALITY_REQUIRED (0x0d)              -- new in LDAPv3
            LDAP_NO_SUCH_ATTRIBUTE (0x10)
            LDAP_UNDEFINED_TYPE (0x11)
            LDAP_INAPPROPRIATE_MATCHING (0x12)
            LDAP_CONSTRAINT_VIOLATION (0x13)
            LDAP_TYPE_OR_VALUE_EXISTS (0x14)
            LDAP_INVALID_SYNTAX (0x15)
            LDAP_NO_SUCH_OBJECT (0x20)
            LDAP_ALIAS_PROBLEM (0x21)
            LDAP_INVALID_DN_SYNTAX (0x22)
            LDAP_IS_LEAF (0x23)                               -- not used in
LDAPv3
            LDAP_ALIAS_DEREF_PROBLEM (0x24)
            LDAP_INAPPROPRIATE_AUTH (0x30)
            LDAP_INVALID_CREDENTIALS (0x31)
            LDAP_INSUFFICIENT_ACCESS (0x32)
            LDAP_BUSY (0x33)
            LDAP_UNAVAILABLE (0x34)
            LDAP_UNWILLING_TO_PERFORM (0x35)
            LDAP_LOOP_DETECT (0x36)
            LDAP_NAMING_VIOLATION (0x40)
            LDAP_OBJECT_CLASS_VIOLATION (0x41)
            LDAP_NOT_ALLOWED_ON_NONLEAF (0x42)
            LDAP_NOT_ALLOWED_ON_RDN (0x43)
            LDAP_ALREADY_EXISTS (0x44)
            LDAP_NO_OBJECT_CLASS_MODS (0x45)
            LDAP_RESULTS_TOO_LARGE (0x46)
            LDAP_AFFECTS_MULTIPLE_DSAS (0x47)                 -- new in LDAPv3
            LDAP_OTHER (0x50)
```

LDAP_SERVER_DOWN (0x51)

```
LDAP_LOCAL_ERROR (0x52)
LDAP_ENCODING_ERROR (0x53)
LDAP_DECODING_ERROR (0x54)
LDAP_TIMEOUT (0x55)
LDAP_AUTH_UNKNOWN (0x56)
LDAP_FILTER_ERROR (0x57)
LDAP_USER_CANCELLED (0x58)
LDAP_PARAM_ERROR (0x59)
LDAP_NO_MEMORY (0x5a)
LDAP_CONNECT_ERROR (0x5b)
LDAP_NOT_SUPPORTED (0x5c)
LDAP_CONTROL_NOT_FOUND (0x5d)
LDAP_NO_RESULTS_RETURNED (0x5e)
LDAP_MORE_RESULTS_TO_RETURN (0x5f)
LDAP_CLIENT_LOOP (0x60)
LDAP_REFERRAL_LIMIT_EXCEEDED (0x61)
```

7. Performing LDAP Operations

This section describes each LDAP operation API call in detail. All functions take a "session handle," a pointer to an LDAP structure containing per-connection information. Many routines return results in an LDAPMessage structure. These structures and others are described as needed below.

7.1. Initializing an LDAP Session

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

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

Use of the following routine is deprecated.

```
LDAP *ldap_open(
        char    *hostname,
        int     portno
);
```

Parameters are:

hostname Contains a space-separated list of hostnames or dotted strings

representing the IP address of hosts running an LDAP server to connect to. Each hostname in the list can include an optional port number which is separated from the host itself with a colon (:) character. The hosts are tried in the order listed, stopping with the first one to which a successful connection is made. Note that only ldap_open() attempts to make the connection before returning to the caller. ldap_init() does not connect to the LDAP server.

portno Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant LDAP_PORT. If a host includes a port number then this parameter is ignored.

ldap_init() and ldap_open() both return a "session handle," a pointer to an opaque structure that should be passed to subsequent calls pertaining to the session. These routines return NULL if the session cannot be initialized in which case the operating system error reporting mechanism can be checked to see why the call failed.

Note that if you connect to an LDAPv2 server, one of the ldap_bind() calls described below must be completed before other operations can be performed on the session. LDAPv3 does not require that a bind operation be completed before other operations can be performed.

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

7.2. LDAP Session Handle Options

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

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

ldap_get_option() is used to access the current value of various session-wide parameters. ldap_set_option() is used to set the value of these parameters.

```
int ldap_get_option(
        LDAP            *ld,
        int             option,
```

```
                void            *outvalue
        );

        int ldap_set_option(
                LDAP            *ld,
                int             option,
                void            *invalue
        );
```

Parameters are:

ld The session handle.

option The name of the option being accessed or set. This parameter should be one of the following constants, which have the indicated meanings. After the constant the actual value of the constant is listed in hexadecimal in parentheses followed by the type of the corresponding outvalue or invalue parameter.

LDAP_OPT_DESC (0x01) int *
: The underlying socket descriptor corresponding to the default LDAP connection.

LDAP_OPT_DEREF (0x02) int *
: Controls how aliases are handled during search. It can have one of the following values: LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03). The LDAP_DEREF_SEARCHING value means aliases should be dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases should be dereferenced when locating the base object but not during the search.

LDAP_OPT_SIZELIMIT (0x03) int *
: A limit on the number of entries to return from a search. A value of zero means no limit.

LDAP_OPT_TIMELIMIT (0x04) int *
: A limit on the number of seconds to spend on a search. A value of zero means no limit

LDAP_OPT_REBIND_FN (0x06) function pointer
: See the discussion of ldap_bind() and friends below.

LDAP_OPT_REBIND_ARG (0x07) void *
: See the discussion of ldap_bind() and friends below.

LDAP_OPT_REFERRALS (0x08) void *

This option controls whether the LDAP library automatically follows referrals returned by LDAP servers or not. It can be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF.

LDAP_OPT_RESTART (0x09) void *
This option controls whether LDAP I/O operations should automatically be restarted if they abort prematurely. It should be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF. This option is useful if an LDAP I/O operation may be interrupted prematurely, for example by a timer going off, or other interrrupt.

LDAP_OPT_PROTOCOL_VERSION (0x11) int *
This option indicates the version of the default LDAP server. It can be one of the constants LDAP_VERSION2 or LDAP_VERSION3. If no version is set the default is LDAP_VERSION2.

LDAP_OPT_SERVER_CONTROLS (0x12) LDAPControl **
A default list of LDAP server controls to be sent with each request. See the Using Controls section below.

LDAP_OPT_CLIENT_CONTROLS (0x13) LDAPControl **
A default list of client controls that affect the LDAP session. See the Using Controls section below.

LDAP_OPT_HOST_NAME (0x30) char **
The host name of the default LDAP server.

LDAP_OPT_ERROR_NUMBER (0x31) int *
The code of the most recent LDAP error that occurred for this session.

LDAP_OPT_ERROR_STRING (0x32) char **
The message returned with the most recent LDAP error that occurred for this session.

outvalue The address of a place to put the value of the option. The actual type of this parameter depends on the setting of the option parameter.

invalue A pointer to the value the option is to be given. The actual type of this parameter depends on the setting of the option parameter. The constants LDAP_OPT_ON and LDAP_OPT_OFF can be given for options that have on or off settings.

7.3. Working with controls

LDAPv3 operations can be extended through the use of controls. Controls may be sent to a server or returned to the client with any LDAP message. These controls are referred to as server controls.

The LDAP API also supports a client-side extension mechanism through the use of client controls. These controls affect the behavior of the LDAP API only and are never sent to a server. A common data structure is used to represent both types of controls:

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

The fields in the ldapcontrol structure have the following meanings:

ldctl_oid The control type, represented as a string.

ldctl_value The data associated with the control (if any).

ldctl_iscritical Indicates whether the control is critical of not. If this field is non-zero, the operation will only be carried out if the control is recognized by the server and/or client.

Some LDAP API calls allocate an ldapcontrol structure or a NULL-terminated array of ldapcontrol structures. The following routines can be used to dispose of a single control or an array of controls:

```
void ldap_control_free( LDAPControl *ctrl );
void ldap_controls_free( LDAPControl **ctrls );
```

A set of controls that affect the entire session can be set using the ldap_set_option() function (see above). A list of controls can also be passed directly to some LDAP API calls such as ldap_search_ext(), in which case any controls set for the session through the use of ldap_set_option() are ignored. Control lists are represented as a NULL-terminated array of pointers to ldapcontrol structures.

Server controls are defined by LDAPv3 protocol extension documents; for example, a control has been proposed to support server-side sorting of search results [7].

No client controls are defined by this document but they may be defined in future revisions or in any document that extends this API.

7.4. Authenticating to the directory

The following functions are used to authenticate an LDAP client to an LDAP directory server.

The ldap_sasl_bind() and ldap_sasl_bind_s() functions can be used to do general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer [8]. The routines both take the dn to bind as, the method to use, as a dotted-string representation of an OID identifying the method, and a struct berval holding the credentials. The special constant value LDAP_SASL_SIMPLE ("") can be passed to request simple authentication, or the simplified routines ldap_simple_bind() or ldap_simple_bind_s() can be used.

```
int ldap_sasl_bind(
        LDAP            *ld,
        char            *dn,
        char            *mechanism,
        struct berval   *cred,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

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

int ldap_simple_bind(
        LDAP            *ld,
        char            *dn,
        char            *passwd
);

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

The use of the following routines is deprecated:

```
int ldap_bind( LDAP *ld, char *dn, char *cred, int method );

int ldap_bind_s( LDAP *ld, char *dn, char *cred, int method );

int ldap_kerberos_bind( LDAP *ld, char *dn );

int ldap_kerberos_bind_s( LDAP *ld, char *dn );
```

Parameters are:

ld	The session handle.
dn	The name of the entry to bind as.
mechanism	Either LDAP_AUTH_SIMPLE_OID to get simple authentication, or a dotted text string representing an OID identifying the SASL method.
cred	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter.
passwd	For ldap_simple_bind(), the password to compare to the entry's userPassword attribute.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_sasl_bind() call succeeds.
servercredp	This result parameter will be set to the credentials returned by the server. This should be freed by calling ldap_If no credentials are returned it will be set to NULL.

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

The ldap_sasl_bind() function initiates an asynchronous bind operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_sasl_bind() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the bind.

The ldap_simple_bind() function initiates a simple asynchronous bind operation and returns the message id of the operation initiated. A subsequent call to ldap_result(), described below, can be used to obtain the result of the bind. In case of error, ldap_simple_bind() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_sasl_bind_s() and ldap_simple_bind_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

Note that if an LDAPv2 server is contacted, no other operations over the connection should be attempted before a bind call has successfully completed.

Subsequent bind calls can be used to re-authenticate over the same connection, and multistep SASL sequences can be accomplished through a sequence of calls to ldap_sasl_bind() or ldap_sasl_bind_s().

7.5. Closing the session

The following functions are used to unbind from the directory, close the connection, and dispose of the session handle.

```
int ldap_unbind( LDAP *ld );

int ldap_unbind_s( LDAP *ld );
```

Parameters are:

ld The session handle.

ldap_unbind() and ldap_unbind_s() both work synchronously, unbinding from the directory, closing the connection, and freeing up the ld structure before returning. There is no server response to an unbind operation. ldap_unbind() returns LDAP_SUCCESS (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to ldap_unbind() or ldap_unbind_s(), the session handle ld is invalid.

7.6. Searching

The following functions are used to search the LDAP directory, returning a requested set of attributes for each entry matched. There are five variations.

```
int ldap_search_ext(
        LDAP            *ld,
        char            *base,
        int             scope,
        char            *filter,
        char            **attrs,
        int             attrsonly,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        struct timeval  *timeoutp,
        int             sizelimit,
        int             *msgidp
);

int ldap_search_ext_s(
        LDAP            *ld,
        char            *base,
        int             scope,
        char            *filter,
        char            **attrs,
        int             attrsonly,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        struct timeval  *timeoutp,
        int             sizelimit,
        LDAPMessage     **res
);

int ldap_search(
        LDAP    *ld,
        char    *base,
        int     scope,
        char    *filter,
        char    **attrs,
        int     attrsonly
);

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

int ldap_search_st(
```

```
                    LDAP            *ld,
                    char            *base,
                    int             scope,
                    char            *filter,
                    char            **attrs,
                    int             attrsonly,
                    struct timeval  *timeout,
                    LDAPMessage     **res
            );
```

Parameters are:

ld	The session handle.
base	The dn of the entry at which to start the search.
scope	One of LDAP_SCOPE_BASE (0x00), LDAP_SCOPE_ONELEVEL (0x01), or LDAP_SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string as described in [3], representing the search filter.
attrs	A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available attributes to be retrieved.
attrsonly	A boolean value that should be zero if both attribute types and values are to be returned, non-zero if only types are wanted.
timeout	For the ldap_search_st() function, this specifies the local search timeout value. For the ldap_search_ext() and ldap_search_ext_s() functions, this specifies both the local search timeout value and the operation time limit that is sent to the server within the search request.
res	For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_search_ext() call succeeds.

There are three options in the session handle ld which potentially affect how the search is performed. They are:

LDAP_OPT_SIZELIMIT
: A limit on the number of entries to return from the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions.

LDAP_OPT_TIMELIMIT
: A limit on the number of seconds to spend on the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions.

LDAP_OPT_DEREF
: One of LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03), specifying how aliases should be handled during the search. The LDAP_DEREF_SEARCHING value means aliases should be dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases should be dereferenced when locating the base object but not during the search.

The ldap_search_ext() function initiates an asynchronous search operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_search_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the results from the search. These results can be parsed using the result parsing routines described in detail later.

Similar to ldap_search_ext(), the ldap_search() function initiates an asynchronous search operation and returns the message id of the operation initiated. As for ldap_search_ext(), a subsequent call to ldap_result(), described below, can be used to obtain the result of the bind. In case of error, ldap_search() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_search_ext_s(), ldap_search_s(), and ldap_search_st() functions all return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them. Entries returned from the search (if any) are contained in the

res parameter. This parameter is opaque to the caller. Entries, attributes, values, etc., should be extracted by calling the parsing routines described below. The results contained in res should be freed when no longer in use by calling ldap_msgfree(), described later.

The ldap_search_ext() and ldap_search_ext_s() functions support LDAPv3 server controls, client controls, and allow varying size and time limits to be easily specified for each search operation. The ldap_search_st() function is identical to ldap_search_s() except that it takes an additional parameter specifying a local timeout for the search.

7.7. Reading an Entry

LDAP does not support a read operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to read, scope set to LDAP_SCOPE_BASE, and filter set to "(objectclass=*)". attrs contains the list of attributes to return.

7.8. Listing the Children of an Entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to list, scope set to LDAP_SCOPE_ONELEVEL, and filter set to "(objectclass=*)". attrs contains the list of attributes to return for each child entry.

7.9. Comparing a Value Against an Entry

The following routines are used to compare a given attribute value assertion against an LDAP entry. There are four variations:

```
int ldap_compare_ext(
        LDAP            *ld,
        char            *dn,
        char            *attr,
        struct berval   *bvalue
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

int ldap_compare_ext_s(
        LDAP            *ld,
        char            *dn,
        char            *attr,
        struct berval   *bvalue,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls
```

```
        );

        int ldap_compare(
                LDAP            *ld,
                char            *dn,
                char            *attr,
                char            *value
        );

        int ldap_compare_s(
                LDAP            *ld,
                char            *dn,
                char            *attr,
                char            *value
        );
```

Parameters are:

ld	The session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
bvalue	The attribute value to compare against those found in the given entry. This parameter is used in the extended routines and is a pointer to a struct berval so it is possible to compare binary values.
value	A string attribute value to compare against, used by the ldap_compare() and ldap_compare_s() functions. Use ldap_compare_ext() or ldap_compare_ext_s() if you need to compare binary values.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_compare_ext() call succeeds.

The ldap_compare_ext() function initiates an asynchronous compare operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_compare_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the compare.

Similar to ldap_compare_ext(), the ldap_compare() function initiates an asynchronous compare operation and returns the message id of the operation initiated. As for ldap_compare_ext(), a subsequent call to ldap_result(), described below, can be used to obtain the result of the bind. In case of error, ldap_compare() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_compare_ext_s() and ldap_compare_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_compare_ext() and ldap_compare_ext_s() functions support LDAPv3 server controls and client controls.

7.10. Modifying an entry

The following routines are used to modify an existing LDAP entry. There are four variations:

```
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_ext(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **mods,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

int ldap_modify_ext_s(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **mods,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls
```

```
        );

        int ldap_modify(
                LDAP            *ld,
                char            *dn,
                LDAPMod         **mods
        );

        int ldap_modify_s(
                LDAP            *ld,
                char            *dn,
                LDAPMod         **mods
        );
```

Parameters are:

ld The session handle.

dn The name of the entry to modify.

mods A NULL-terminated array of modifications to make to the entry.

serverctrls List of LDAP server controls.

clientctrls List of client controls.

msgidp This result parameter will be set to the message id of the request if the ldap_modify_ext() call succeeds.

The fields in the LDAPMod structure have the following meanings:

mod_op The modification operation to perform. It should be one of LDAP_MOD_ADD (0x00), LDAP_MOD_DELETE (0x01), or LDAP_MOD_REPLACE (0x02). This field also indicates the type of values included in the mod_vals union. It is logically ORed with LDAP_MOD_BVALUES (0x80) to select the mod_bvalues form. Otherwise, the mod_values form is used.

mod_type The type of the attribute to modify.

mod_vals The values (if any) to add, delete, or replace. Only one of the mod_values or mod_bvalues variants should be used, selected by ORing the mod_op field with the constant LDAP_MOD_BVALUES. mod_values is a NULL-terminated array of zero-terminated strings and mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

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

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

For LDAP_MOD_REPLACE modifications, the attribute will have the listed values after the modification, having been created if necessary, or removed if the mod_vals field is NULL. All modifications are performed in the order in which they are listed.

The ldap_modify_ext() function initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_modify_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the modify.

Similar to ldap_modify_ext(), the ldap_modify() function initiates an asynchronous modify operation and returns the message id of the operation initiated. As for ldap_modify_ext(), a subsequent call to ldap_result(), described below, can be used to obtain the result of the modify. In case of error, ldap_modify() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_modify_ext_s() and ldap_modify_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_modify_ext() and ldap_modify_ext_s() functions support LDAPv3 server controls and client controls.

7.11. Modifying the Name of an Entry

In LDAPv2, the ldap_modrdn() and ldap_modrdn_s() routines were used to change the name of an LDAP entry. They could only be used to change the least significant component of a name (the RDN or relative distinguished name). LDAPv3 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, and the use of the ldap_modrdn() and ldap_modrdn_s() routines is deprecated.

```
int ldap_rename(
```

```
                LDAP            *ld,
                char            *dn,
                char            *newrdn,
                char            *newparent,
                int             deleteoldrdn,
                LDAPControl     **serverctrls,
                LDAPControl     **clientctrls,
                int             *msgidp

        );
        int ldap_rename_s(
                LDAP            *ld,
                char            *dn,
                char            *newrdn,
                char            *newparent,
                int             deleteoldrdn,
                LDAPControl     **serverctrls,
                LDAPControl     **clientctrls
        );
```

Use of the following routines is deprecated.

```
        int ldap_modrdn(
                LDAP    *ld,
                char    *dn,
                char    *newrdn,
                int     deleteoldrdn
        );
        int ldap_modrdn_s(
                LDAP    *ld,
                char    *dn,
                char    *newrdn,
                int     deleteoldrdn
        );
```

Parameters are:

ld The session handle.

dn The name of the entry whose DN is to be changed.

newrdn The new RDN to give the entry.

newparent The new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is changed. The root DN may be specified by passing a zero length string, "". The newparent parameter should always be NULL when using version 2 of the LDAP protocol; otherwise the server's

behavior is undefined.

deleteoldrdn This parameter only has meaning on the rename routines if newrdn is different than the old RDN. It is a boolean value, if non-zero indicating that the old RDN value(s) should be removed, if zero indicating that the old RDN value(s) should be retained as non-distinguished values of the entry.

serverctrls List of LDAP server controls.

clientctrls List of client controls.

msgidp This result parameter will be set to the message id of the request if the ldap_rename() call succeeds.

The ldap_rename() function initiates an asynchronous modify DN operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_rename() places the DN message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the rename.

The synchronous ldap_rename_s() returns the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_rename() and ldap_rename_s() functions both support LDAPv3 server controls and client controls.

7.12. Adding an entry

The following functions are used to add entries to the LDAP directory. There are four variations:

```
int ldap_add_ext(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **attrs,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);
```

```
int ldap_add_ext_s(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **attrs,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls
);

int ldap_add(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **attrs
);

int ldap_add_s(
        LDAP            *ld,
        char            *dn,
        LDAPMod         **attrs
);
```

Parameters are:

ld — The session handle.

dn — The name of the entry to add.

attrs — The entry's attributes, specified using the LDAPMod structure defined for ldap_modify(). The mod_type and mod_vals fields should be filled in. The mod_op field is ignored unless ORed with the constant LDAP_MOD_BVALUES, used to select the mod_bvalues case of the mod_vals union.

serverctrls — List of LDAP server controls.

clientctrls — List of client controls.

msgidp — This result parameter will be set to the message id of the request if the ldap_add_ext() call succeeds.

Note that the parent of the entry being added must already exist or the parent must be empty (i.e., equal to the root DN) for an add to succeed.

The ldap_add_ext() function initiates an asynchronous add operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_add_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below,

can be used to obtain the result of the add.

Similar to ldap_add_ext(), the ldap_add() function initiates an asynchronous add operation and returns the message id of the operation initiated. As for ldap_add_ext(), a subsequent call to ldap_result(), described below, can be used to obtain the result of the add. In case of error, ldap_add() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_add_ext_s() and ldap_add_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_add_ext() and ldap_add_ext_s() functions support LDAPv3 server controls and client controls.

7.13. Deleting an entry

The following functions are used to delete a leaf entry from the LDAP directory. There are four variations:

```
int ldap_delete_ext(
        LDAP            *ld,
        char            *dn,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

int ldap_delete_ext_s(
        LDAP            *ld,
        char            *dn,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls
);

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

int ldap_delete_s(
        LDAP            *ld,
        char            *dn
```

```
        );
```

Parameters are:

ld	The session handle.
dn	The name of the entry to delete.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_delete_ext() call succeeds.

Note that the entry to delete must be a leaf entry (i.e., it must have no children). Deletion of entire subtrees in a single operation is not supported by LDAP.

The ldap_delete_ext() function initiates an asynchronous delete operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_delete_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the delete.

Similar to ldap_delete_ext(), the ldap_delete() function initiates an asynchronous delete operation and returns the message id of the operation initiated. As for ldap_delete_ext(), a subsequent call to ldap_result(), described below, can be used to obtain the result of the delete. In case of error, ldap_delete() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_delete_ext_s() and ldap_delete_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_delete_ext() and ldap_delete_ext_s() functions support LDAPv3 server controls and client controls.

7.14. Extended Operations

The ldap_extended_operation() and ldap_extended_operation_s() routines allow extended LDAP operations to be passed to the server, providing a

general protocol extensibility mechanism.

```
int ldap_extended_operation(
        LDAP            *ld,
        char            *exoid,
        struct berval   *exdata,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

int ldap_extended_operation_s(
        LDAP            *ld,
        char            *exoid,
        struct berval   *exdata,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        char            **retoidp,
        struct berval   **retdatap
);
```

Parameters are:

ld The session handle.

requestoid The dotted-OID text string naming the request.

requestdata The arbitrary data required by the operation (if NULL, no data is sent to the server).

serverctrls List of LDAP server controls.

clientctrls List of client controls.

msgidp This result parameter will be set to the message id of the request if the ldap_extended_operation() call succeeds.

retoidp Pointer to a character string that will be set to an allocated, dotted-OID text string returned by the server. This string should be disposed of using the ldap_memfree() function. If no OID was returned, *retoidp is set to NULL.

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

The ldap_extended_operation() function initiates an asynchronous

extended operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_extended_operation() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, can be used to obtain the result of the extended operation which can be passed to ldap_parse_extended_result() to obtain the OID and data contained in the response.

The synchronous ldap_extended_operation_s() function returns the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not. See the section below on error handling for more information about possible errors and how to interpret them. The retoid and retdata parameters are filled in with the OID and data from the response. If no OID or data was returned, these parameters are set to NULL.

The ldap_extended_operation() and ldap_extended_operation_s() functions both support LDAPv3 server controls and client controls.

8. Abandoning An Operation

The following calls are used to abandon an operation in progress:

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

int ldap_abandon(
        LDAP            *ld,
        int             msgid
);
```

ld The session handle.

msgid The message id of the request to be abandoned.

serverctrls List of LDAP server controls.

clientctrls List of client controls.

ldap_abandon_ext() abandons the operation with message id msgid and returns the constant LDAP_SUCCESS if the abandon was successful or

another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them.

ldap_abandon() is identical to ldap_abandon_ext() except that it returns zero if the abandon was successful, -1 otherwise and does not support LDAPv3 server controls or client controls.

After a successful call to ldap_abandon() or ldap_abandon_ext(), results with the given message id are never returned from a subsequent call to ldap_result(). There is no server response to LDAP abandon operations.

9. Obtaining Results and Peeking Inside LDAP Messages

ldap_result() is used to obtain the result of a previous asynchronously initiated operation. Note that depending on how it is called, ldap_result() may actually return a list or "chain" of messages.

ldap_msgfree() frees the results obtained from a previous call to ldap_result(), or a synchronous search routine.

ldap_msgtype() returns the type of an LDAP message. ldap_msgid() returns the message ID of an LDAP message.

```
int ldap_result(
        LDAP            *ld,
        int             msgid,
        int             all,
        struct timeval  *timeout,
        LDAPMessage     **res
);

int ldap_msgfree( LDAPMessage *res );

int ldap_msgtype( LDAPMessage *res );

int ldap_msgid( LDAPMessage *res );
```

Parameters are:

ld The session handle.

msgid The message id of the operation whose results are to be returned, or the constant LDAP_RES_ANY (-1) if any result is desired.

all Specifies how many messages will be retrieved in a single call to ldap_result(). This parameter only has meaning for search

results. Pass the constant LDAP_MSG_ONE (0x00) to retrieve one message at a time. Pass LDAP_MSG_ALL (0x01) to request that all results of a search be received before returning all results in a single chain. Pass LDAP_MSG_RECEIVED (0x02) to indicate that all results retrieved so far should be returned in the result chain.

timeout A timeout specifying how long to wait for results to be returned. A NULL value causes ldap_result() to block until results are available. A timeout value of zero seconds specifies a polling behavior.

res For ldap_result(), a result parameter that will contain the result(s) of the operation. For ldap_msgfree(), the result chain to be freed, obtained from a previous call to ldap_result(), ldap_search_s(), or ldap_search_st().

Upon successful completion, ldap_result() returns the type of the first result returned in the res parameter. This will be one of the following constants.

```
LDAP_RES_BIND (0x61)
LDAP_RES_SEARCH_ENTRY (0x64)
LDAP_RES_SEARCH_REFERENCE (0x73)      -- new in LDAPv3
LDAP_RES_SEARCH_RESULT (0x65)
LDAP_RES_MODIFY (0x67)
LDAP_RES_ADD (0x69)
LDAP_RES_DELETE (0x6B)
LDAP_RES_MODDN (0x6D)
LDAP_RES_COMPARE (0x6F)
LDAP_RES_EXTENDED (0x78)              -- new in LDAPv3
```

ldap_result() returns 0 if the timeout expired and -1 if an error occurs, in which case the error parameters of the LDAP session handle will be set accordingly.

ldap_msgfree() frees the result structure pointed to by res and returns the type of the message it freed.

ldap_msgtype() returns the type of the LDAP message it is passed as a parameter. The type will be one of the types listed above, or -1 on error.

ldap_msgid() returns the message ID associated with the LDAP message passed as a parameter.

10. Handling Errors and Parsing Results

The following calls are used to extract information from results and handle errors returned by other LDAP API routines.

```
int ldap_parse_result(
        LDAP            *ld,
        LDAPMessage     *res,
        int             *errcodep,
        char            **matcheddnp,
        char            **errmsgp,
        char            ***referralsp,
        LDAPControl     ***serverctrlsp,
        int             freeit
);

int ldap_parse_sasl_bind_result(
        LDAP            *ld,
        LDAPMessage     *res,
        struct berval   **servercredp,
        int             freeit
);

int ldap_parse_extended_result(
        LDAP            *ld,
        LDAPMessage     *res,
        char            **resultoidp,
        struct berval   **resultdata,
        int             freeit
);

char *ldap_err2string( int err );
```

The use of the following routines is deprecated.

```
int ldap_result2error(
        LDAP            *ld,
        LDAPMessage     *res,
        int             freeit
);

void ldap_perror( LDAP *ld, char *msg );
```

Parameters are:

ld The session handle.

res The result of an LDAP operation as returned by

ldap_result() or one of the synchronous API operation calls.

errcodep This result parameter will be filled in with the LDAP error code field from the LDAPResult message. This is the indication from the server of the outcome of the operation. NULL may be passed to ignore this field.

matcheddnp In the case of a return of LDAP_NO_SUCH_OBJECT, this result parameter will be filled in with a DN indicating how much of the name in the request was recognized. NULL may be passed to ignore this field. The matched DN string should be freed by calling ldap_memfree() which is described later in this document.

errmsgp This result parameter will be filled in with the contents of the error message field from the LDAPResult message. The error message string should be freed by calling ldap_memfree() which is described later in this document. NULL may be passed to ignore this field.

referralsp This result parameter will be filled in with the contents of the referrals field from the LDAPResult message, indicating zero or more alternate LDAP servers where the request should be retried. The referrals array should be freed by calling ldap_value_free() which is described later in this document. NULL may be passed to ignore this field.

serverctrlsp This result parameter will be filled in with an allocated array of controls copied out of the LDAPResult message. The control array should be freed by calling ldap_controls_free() which was described earlier.

freeit A boolean that determines whether the res parameter is disposed of or not. Pass any non-zero value to have these routines free res after extracting the requested information. This is provided as a convenience; you can also use ldap_msgfree() to free the result later.

servercredp For SASL bind results, this result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated berval structure is returned that should be disposed of by calling ldap_ber_free(). NULL may be passed to ignore this field.

resultoidp For extended results, this result parameter will be filled in with the dotted-OID text representation of the name of the extended operation response. This string should be

disposed of by calling ldap_memfree(). NULL may be passed to ignore this field.

resultdatap For extended results, this result parameter will be filled in with a pointer to a struct berval containing the data in the extended operation response. It should be disposed of by calling ldap_ber_free(). NULL may be passed to ignore this field.

err For ldap_err2string(), an LDAP error code, as returned by ldap_result2error() or another LDAP API call.

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

All of the ldap_parse_*_result() routines skip over messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE when looking for a result message to parse. They return the constant LDAP_SUCCESS if the result was successfully parsed and another LDAP error code if not. Note that the LDAP error code that indicates the outcome of the operation performed by the server is placed in the errcodep ldap_parse_result() parameter.

ldap_err2string() is used to convert a numeric LDAP error code, as returned by one of the ldap_parse_*_result() routines, or one of the synchronous API operation calls, into an informative NULL-terminated character string message describing the error. It returns a pointer to static data.

11. Stepping Through a List of Results

The ldap_first_message() and ldap_next_message() routines are used to step through the list of messages in a result chain returned by ldap_result(). For search operations, the result chain may actually include referral messages, entry messages, and result messages. ldap_count_messages() is used to count the number of messages returned. The ldap_msgtype() function, described above, can be used to distinguish between the different message types.

```
LDAPMessage *ldap_first_message( LDAP *ld, LDAPMessage *res );

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

int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters are:

ld The session handle.

res The result chain, as obtained by a call to one of the synchronous search routines or ldap_result().

msg The message returned by a previous call to ldap_first_message() or ldap_next_message().

ldap_first_message() and ldap_next_message() will return NULL when no more messages exist in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle ld will be set to indicate the error.

ldap_count_messages() returns the number of messages contained in a chain of results. It can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

12. Parsing Search Results

The following calls are used to parse the entries and references returned by ldap_search() and friends. These results are returned in an opaque structure that should only be accessed by calling the routines described below. Routines are provided to step through the entries and references returned, step through the attributes of an entry, retrieve the name of an entry, and retrieve the values associated with a given attribute in an entry.

12.1. Stepping Through a List of Entries

The ldap_first_entry() and ldap_next_entry() routines are used to step through and retrieve the list of entries from a search result chain. The ldap_first_reference() and ldap_next_reference() routines are used to step through and retrieve the list of continuation references from a search result chain. ldap_count_entries() is used to count the number of entries returned. ldap_count_references() is used to count the number of references returned.

```
LDAPMessage *ldap_first_entry( LDAP *ld, LDAPMessage *res );

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

LDAPMessage *ldap_first_reference( LDAP *ld, LDAPMessage *res );
```

```
LDAPMessage *ldap_next_reference( LDAP *ld, LDAPMessage *ref );

int ldap_count_entries( LDAP *ld, LDAPMessage *res );

int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters are:

ld The session handle.

res The search result, as obtained by a call to one of the synchronous search routines or ldap_result().

entry The entry returned by a previous call to ldap_first_entry() or ldap_next_entry().

ldap_first_entry() and ldap_next_entry() will return NULL when no more entries or references exist in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle ld will be set to indicate the error.

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

ldap_count_references() returns the number of references contained in a chain of search results. It can also be used to count the number of references that remain in a chain.

12.2. Stepping Through the Attributes of an Entry

The ldap_first_attribute() and ldap_next_attribute() calls are used to step through the list of attribute types returned with an entry.

```
char *ldap_first_attribute(
        LDAP            *ld,
        LDAPMessage     *entry,
        BerElement      **ptr
);

char *ldap_next_attribute(
        LDAP            *ld,
        LDAPMessage     *entry,
        BerElement      *ptr
```

```
        );

        void ldap_memfree( char *mem );
```

Parameters are:

ld The session handle.

entry The entry whose attributes are to be stepped through, as returned by ldap_first_entry() or ldap_next_entry().

ptr In ldap_first_attribute(), the address of a pointer used internally to keep track of the current position in the entry. In ldap_next_attribute(), the pointer returned by a previous call to ldap_first_attribute().

mem A pointer to memory allocated by the LDAP library, such as the attribute names returned by ldap_first_attribute() and ldap_next_attribute, or the DN returned by ldap_get_dn().

ldap_first_attribute() and ldap_next_attribute() will return NULL when the end of the attributes is reached, or if there is an error, in which case the error parameters in the session handle ld will be set to indicate the error.

Both routines return a pointer to an allocated buffer containing the current attribute name. This should be freed when no longer in use by calling ldap_memfree().

ldap_first_attribute() will allocate and return in ptr a pointer to a BerElement used to keep track of the current position. This pointer should be passed in subsequent calls to ldap_next_attribute() to step through the entry's attributes. After a set of calls to ldap_first_attribute() and ldap_next_attibute(), if ptr is non-NULL, it should be freed by calling ldap_ber_free(ptr, 0). Note that it is very important to pass the second parameter as 0 (zero) in this call.

The attribute names returned are suitable for passing in a call to ldap_get_values() and friends to retrieve the associated values.

12.3. Retrieving the Values of an Attribute

ldap_get_values() and ldap_get_values_len() are used to retrieve the values of a given attribute from an entry. ldap_count_values() and ldap_count_values_len() are used to count the returned values. ldap_value_free() and ldap_value_free_len() are used to free the values.

```
char **ldap_get_values(
        LDAP            *ld,
        LDAPMessage     *entry,
        char            *attr
);

struct berval **ldap_get_values_len(
        LDAP            *ld,
        LDAPMessage     *entry,
        char            *attr
);

int ldap_count_values( char **vals );

int ldap_count_values_len( struct berval **vals );

int ldap_value_free( char **vals );

int ldap_value_free_len( struct berval **vals );
```

Parameters are:

ld The session handle.

entry The entry from which to retrieve values, as returned by ldap_first_entry() or ldap_next_entry().

attr The attribute whose values are to be retrieved, as returned by ldap_first_attribute() or ldap_next_attribute(), or a caller-supplied string (e.g., "mail").

vals The values returned by a previous call to ldap_get_values() or ldap_get_values_len().

Two forms of the various calls are provided. The first form is only suitable for use with non-binary character string data. The second _len form is used with any kind of data.

Note that the values returned are dynamically allocated and should be freed by calling either ldap_value_free() or ldap_value_free_len() when no longer in use.

12.4. Retrieving the name of an entry

ldap_get_dn() is used to retrieve the name of an entry. ldap_explode_dn() and ldap_explode_rdn() are used to break up a name into its component parts. ldap_dn2ufn() is used to convert the name into

a more "user friendly" format.

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

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

char **ldap_explode_rdn( char *rdn, int notypes );

char *ldap_dn2ufn( char *dn );
```

Parameters are:

ld The session handle.

entry The entry whose name is to be retrieved, as returned by ldap_first_entry() or ldap_next_entry().

dn The dn to explode, such as returned by ldap_get_dn().

rdn The rdn to explode, such as returned in the components of the array returned by ldap_explode_dn().

notypes A boolean parameter, if non-zero indicating that the dn or rdn components should have their type information stripped off (i.e., "cn=Babs" would become "Babs").

ldap_get_dn() will return NULL if there is some error parsing the dn, setting error parameters in the session handle ld to indicate the error. It returns a pointer to malloc'ed space that the caller should free by calling ldap_memfree() when it is no longer in use. Note the format of the DNs returned is given by [4].

ldap_explode_dn() returns a NULL-terminated char * array containing the RDN components of the DN supplied, with or without types as indicated by the notypes parameter. The array returned should be freed when it is no longer in use by calling ldap_value_free().

ldap_explode_rdn() returns a NULL-terminated char * array containing the components of the RDN supplied, with or without types as indicated by the notypes parameter. The array returned should be freed when it is no longer in use by calling ldap_value_free().

ldap_dn2ufn() converts the DN into the user friendly format described in [5]. The UFN returned is malloc'ed space that should be freed by a call to ldap_memfree() when no longer in use.

13. Encoded ASN.1 Value Manipulation

This section describes routines which may be used to encode and decode BER-encoded ASN.1 values, which are often used inside of control and extension values.

With the exceptions of two new functions ber_flatten() and ber_init(), these functions are compatible with the University of Michigan LDAP 3.3 implementation of BER.

13.1. General

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

A struct berval contains a sequence of bytes and an indication of its length. The bv_val is not null terminated. bv_len must always be a nonnegative number. Applications may allocate their own berval structures.

```
typedef struct berelement {
        /* opaque */
} BerElement;
```

The BerElement structure contains not only a copy of the encoded value, but also state information used in encoding or decoding. Applications cannot allocate their own BerElement structures. The internal state is neither thread-specific nor locked, so two threads should not manipulate the same BerElement value simultaneously.

A single BerElement value cannot be used for both encoding and decoding.

```
void ber_bvfree ( struct berval *bv);
```

ber_bvfree() frees a berval returned from this API. Both the bv->bv_val string and the berval itself are freed. Applications should not use ber_bvfree() with bervals which the application has allocated.

```
void ber_bvecfree ( struct berval **bv );
```

ber_bvecfree() frees an array of bervals returned from this API. Each of the bervals in the array are freed using ber_bvfree(), then the array itself is freed.

```
struct berval *ber_bvdup (struct berval *bv );
```

ber_bvdup() returns a copy of a berval. The bv_val field in the returned berval points to a different area of memory as the bv_val field in the argument berval. The null pointer is returned on error (e.g. out of memory).

```
void ber_free ( BerElement *ber, int fbuf );
```

ber_free() frees a BerElement which is returned from the API calls ber_alloc_t() or ber_init(). Each BerElement must be freed by the caller. The second argument fbuf should always be set to 1.

13.2. Encoding

```
BerElement *ber_alloc_t(int options);
```

ber_alloc_t() constructs and returns BerElement. The null pointer is returned on error. The options field contains a bitwise-or of options which are to be used when generating the encoding of this BerElement. One option is defined and must always be supplied:

```
#define LBER_USE_DER 0x01
```

When this option is present, lengths will always be encoded in the minimum number of octets. Note that this option does not cause values of sets and sequences to be rearranged in tag and byte order, so these functions are not suitable for generating DER output as defined in X.509 and X.680.

Unrecognized option bits are ignored.

The BerElement returned by ber_alloc_t() is initially empty. Calls to ber_printf() will append bytes to the end of the ber_alloc_t().

```
int ber_printf(BerElement *ber, char *fmt, ... )
```

The ber_printf() routine is used to encode a BER element in much the same way that sprintf() works. One important difference, though, is that state information is kept in the ber argument so that multiple calls can be made to ber_printf() to append to the end of the BER element. ber must be a pointer to a BerElement returned by ber_alloc_t(). ber_printf() interprets and formats its arguments according to the format string fmt. ber_printf() returns -1 if there is an error during encoding. As with sprintf(), each character in fmt refers to an argument to ber_printf().

The format string can contain the following format characters:

't' Tag. The next argument is an int specifying the tag to override the next element to be written to the ber. This works across calls. The int value must contain the tag class, constructed bit, and tag value. The tag value must fit in a single octet (tag value is less than 32). For example, a tag of "[3]" for a constructed type is 0xA3.

'b' Boolean. The next argument is an int, containing either 0 for FALSE or 0xff for TRUE. A boolean element is output. If this format character is not preceded by the 't' format modifier, the tag 0x01 is used for the element.

'i' Integer. The next argument is an int, containing the integer in the host's byte order. An integer element is output. If this format character is not preceded by the 't' format modifier, the tag 0x02 is used for the element.

'X' Bitstring. The next two arguments are a char * pointer to the start of the bitstring, followed by an int containing the number of bits in the bitstring. A bitstring element is output, in primitive form. If this format character is not preceded by the 't' format modifier, the tag 0x03 is used for the element.

'n' Null. No argument is required. An ASN.1 NULL element is output. If this format character is not preceded by the 't' format modifier, the tag 0x05 is used for the element.

'o' Octet string. The next two arguments are a char *, followed by an int with the length of the string. The string may contain null bytes and need not by null-terminated. An octet string element is output, in primitive form. If this format character is not preceded by the 't' format modifier, the tag 0x04 is used for the element.

's' Octet string. The next argument is a char * pointing to a null-terminated string. An octet string element in primitive form is output, which does not include the trailing ' ' byte. If this format character is not preceded by the 't' format modifier, the tag 0x04 is used for the element.

'v' Several octet strings. The next argument is a char **, an array of char * pointers to null-terminated strings. The last element in the array must be a null pointer. The octet strings do not include the trailing SEQUENCE OF octet strings. The 't' format modifier cannot be used with this format character.

'V' Several octet strings. A null-terminated array of berval *'s is supplied. Note that a construct like '{V}' is required to get an

actual SEQUENCE OF octet strings. The 't' format modifier cannot be used with this format character.

'{' Begin sequence. No argument is required. If this format character is not preceded by the 't' format modifier, the tag 0x30 is used.

'}' End sequence. No argument is required. The 't' format modifier cannot be used with this format character.

'[' Begin set. No argument is required. If this format character is not preceded by the 't' format modifier, the tag 0x31 is used.

']' End set. No argument is required. The 't' format modifier cannot be used with this format character.

Each use of a '{' format character must be matched by a '}' character, either later in the format string, or in the format string of a subsequent call to ber_printf() for that BerElement. The same applies to the '[' and

Sequences and sets nest, and implementations of this API must maintain internal state to be able to properly calculate the lengths.

```
int ber_flatten (BerElement *ber, struct berval **bvPtr);
```

The ber_flatten routine allocates a struct berval whose contents are a BER encoding taken from the ber argument. The bvPtr pointer points to the returned berval, which must be freed using ber_bvfree(). This routine returns 0 on success and -1 on error.

The ber_flatten API call is not present in U-M LDAP 3.3.

The use of ber_flatten on a BerElement in which all '{' and '}' format modifiers have not been properly matched can result in a berval whose contents are not a valid BER encoding.

13.3. Encoding Example

The following is an example of encoding the following ASN.1 data type:

```
Example1Request ::= SEQUENCE {
        s     OCTET STRING, -- must be printable
        val1  INTEGER,
        val2  [0] INTEGER DEFAULT 0
}
```

```
int encode_example1(char *s,int val1,int val2,struct berval **bvPtr)
{
     BerElement *ber;
     int rc;

     ber = ber_alloc_t(LBER_USE_DER);

     if (ber == NULL) return -1;

     if (ber_printf(ber,"{si",s,val1) == -1) {
             ber_free(ber,1);
             return -1;
     }

     if (val2 != 0) {
             if (ber_printf(ber,"ti",0x80,val2) == -1) {
                     ber_free(ber,1);
                     return -1;
             }
     }

     if (ber_printf(ber,"}") == -1) {
             ber_free(ber,1);
             return -1;
     }

     rc = ber_flatten(ber,bvPtr);
     ber_free(ber,1);
     return -1;
}
```

13.4. Decoding

The following two symbols are available to applications.

```
#define LBER_ERROR   0xffffffffL
#define LBER_DEFAULT 0xffffffffL

BerElement *ber_init (struct berval *bv);
```

The ber_init functions construct BerElement and returns a new BerElement containing a copy of the data in the bv argument. ber_init returns the null pointer on error.

```
unsigned long ber_scanf (BerElement *ber, char *fmt, ... );
```

The ber_scanf() routine is used to decode a BER element in much the same

way that sscanf() works. One important difference, though, is that some state information is kept with the ber argument so that multiple calls can be made to ber_scanf() to sequentially read from the BER element. The ber argument must be a pointer to a BerElement returned by ber_init(). ber_scanf interprets the bytes according to the format string fmt, and stores the results in its additional arguments. ber_scanf() returns LBER_ERROR on error, and a nonnegative number on success.

The format string contains conversion specifications which are used to direct the interpretation of the BER element. The format string can contain the following characters:

'a' Octet string. A char ** argument should be supplied. Memory is allocated, filled with the contents of the octet string, null-terminated, and the pointer to the string is stored in the argument. The returned value must be freed using ldap_memfree. The tag of the element must indicate the primitive form (constructed strings are not supported) but is otherwise ignored and discarded during the decoding. This format cannot be used with octet strings which could contain null bytes.

'O' Octet string. A struct berval ** argument should be supplied, which upon return points to a allocated struct berval containing the octet string and its length. ber_bvfree() must be called to free the allocated memory. The tag of the element must indicate the primitive form (constructed strings are not supported) but is otherwise ignored during the decoding.

'b' Boolean. A pointer to an int should be supplied. The int value stored will be 0 for FALSE or nonzero for TRUE. The tag of the element must indicate the primitive form but is otherwise ignored during the decoding.

'i' Integer. A pointer to an int should be supplied. The int value stored will be in host byte order. The tag of the element must indicate the primitive form but is otherwise ignored during the decoding. ber_scanf() will return an error if the integer cannot be stored in an int.

'B' Bitstring. A char ** argument should be supplied which will point to the allocated bits, followed by an unsigned long * argument, which will point to the length (in bits) of the bitstring returned. ldap_memfree must be called to free the bitstring. The tag of the element must indicate the primitive form (constructed bitstrings are not supported) but is otherwise ignored during the decoding.

'n' Null. No argument is required. The element is simply skipped if it is recognized as a zero-length element. The tag is ignored.

'v' Several octet strings. A char *** argument should be supplied, which upon return points to a allocated null-terminated array of char *'s containing the octet strings. NULL is stored if the sequence is empty. ldap_memfree must be called to free each element of the array and the array itself. The tag of the sequence and of the octet strings are ignored.

'V' Several octet strings (which could contain null bytes). A struct berval *** should be supplied, which upon return points to a allocated null-terminated array of struct berval *'s containing the octet strings and their lengths. NULL is stored if the sequence is empty. ber_bvecfree() can be called to free the allocated memory. The tag of the sequence and of the octet strings are ignored.

'x' Skip element. The next element is skipped. No argument is required.

'{' Begin sequence. No argument is required. The initial sequence tag and length are skipped.

'}' End sequence. No argument is required.

'[' Begin set. No argument is required. The initial set tag and length are skipped.

']' End set. No argument is required.

```
unsigned long ber_peek_tag (BerElement *ber, unsigned long *lenPtr);
```

ber_peek_tag() returns the tag of the next element to be parsed in the BerElement argument. The length of this element is stored in the *lenPtr argument. LBER_DEFAULT is returned if there is no further data to be read. The ber argument is not modified.

```
unsigned long ber_skip_tag (BerElement *ber, unsigned long *lenPtr);
```

ber_skip_tag() is similar to ber_peek_tag(), except that the state pointer in the BerElement argument is advanced past the first tag and length, and is pointed to the value part of the next element. This routine should only be used with constructed types and situations when a BER encoding is used as the value of an OCTET STRING. The length of the value is stored in *lenPtr.

```
unsigned long ber_first_element(BerElement *ber,
        unsigned long *lenPtr, char **opaquePtr);

unsigned long ber_next_element  (BerElement *ber,
        unsigned long *lenPtr, char *opaque);
```

ber_first_element() and ber_next_element() are used to traverse a SET, SET OF, SEQUENCE or SEQUENCE OF data value. ber_first_element() calls ber_skip_tag(), stores internal information in *lenPtr and *opaquePtr, and calls ber_peek_tag() for the first element inside the constructed value. LBER_DEFAULT is returned if the constructed value is empty. ber_next_element() positions the state at the start of the next element in the constructed type. LBER_DEFAULT is returned if there are no further values.

The len and opaque values should not be used by applications other than as arguments to ber_next_element(), as shown in the example below.

13.5. Decoding Example

The following is an example of decoding an ASN.1 data type:

```
Example2Request ::= SEQUENCE {
     dn    OCTET STRING, -- must be printable
     scope ENUMERATED { b (0), s (1), w (2) },
     ali   ENUMERATED { n (0), s (1), f (2), a (3) },
     size  INTEGER,
     time  INTEGER,
     tonly BOOLEAN,
     attrs SEQUENCE OF OCTET STRING, -- must be printable
     [0] SEQUENCE OF SEQUENCE {
        type  OCTET STRING -- must be printable,
        crit  BOOLEAN DEFAULT FALSE,
        value OCTET STRING
     } OPTIONAL }

#define LDAP_TAG_CONTROL_LIST 0xA0L /* context specific cons 0 */

int decode_example2(struct berval *bv)
{
     BerElement *ber;
     unsigned long len;
     int scope, ali, size, time, tonly;
     char *dn = NULL, **attrs = NULL;
     int res,i,rc = 0;

     ber = ber_init(bv);
```

```
        if (ber == NULL) {
                printf("ERROR ber_init failed0);
                return -1;
        }

        res = ber_scanf(ber,"{aiiiiib{v}",&dn,&scope,&ali,
                        &size,&time,&tonly,&attrs);

        if (res == -1) {
                printf("ERROR ber_scanf failed0);
                ber_free(ber,1);
                return -1;
        }

        /* *** use dn */
        ldap_memfree(dn);

        for (i = 0; attrs != NULL && attrs[i] != NULL; i++) {
                /* *** use attrs[i] */
                ldap_memfree(attrs[i]);
        }
        ldap_memfree(attrs);

        if (ber_peek_tag(ber,&len) == LDAP_TAG_CONTROL_LIST) {
                char *opaque;
                unsigned long tag;

                for (tag = ber_first_element(ber,&len,&opaque);
                     tag != LBER_DEFAULT;
                     tag = ber_next_element (ber,&len,opaque)) {

                        unsigned long ttag, tlen;
                        char *type;
                        int crit;
                        struct berval *value;

                        if (ber_scanf(ber,"{a",&type) == LBER_ERROR) {
                                printf("ERROR cannot parse type0);
                                break;
                        }
                        /* *** use type */
                        ldap_memfree(type);

                        ttag = ber_peek_tag(ber,&tlen);
                        if (ttag == 0x01) {  /* boolean */
                                if (ber_scanf(ber,"b",
                                              &crit) == LBER_ERROR) {
                                        printf("ERROR cannot parse crit0);
```

```
                                        rc = -1;
                                        break;
                                }
                        } else if (ttag == 0x04) { /* octet string */
                                crit = 0;
                        } else {
                                printf("ERROR extra field in controls0);
                                break;
                        }

                        if (ber_scanf(ber,"O}",&value) == LBER_ERROR) {
                                printf("ERROR cannot parse value0);
                                rc = -1;
                                break;
                        }
                        /* *** use value */
                        ldap_bvfree(value);
                }
        }

        ber_scanf(ber,"}");

        ber_free(ber,1);

        return rc;
    }
```

14. Security Considerations

LDAPv2 supports security through protocol-level authentication using clear-text passwords. LDAPv3 adds support for SASL [8] (Simple Authentication Security Layer) methods. LDAPv3 also supports operation over a secure transport layer using Transport Layer Security TLS [8]. Readers are referred to the protocol documents for discussion of related security considerations.

Implementations of this API should be cautious when handling authentication credentials. In particular, keeping long-lived copies of credentials without the application's knowledge is discouraged.

15. Acknowledgements

Many members of the IETF ASID working group as well as members of the Internet at large have provided useful comments and suggestions that have been incorporated into this revision.

This original material upon which this revision is based was based upon work supported by the National Science Foundation under Grant No. NCR-9416667.

16. Bibliography

[1] The Directory: Selected Attribute Syntaxes. CCITT, Recommendation X.520.

[2] M. Wahl, A. Coulbeck, T. Howes, S. Kille, W. Yeong, C. Robbins, "Lightweight Directory Access Protocol Attribute Syntax Definitions", INTERNET-DRAFT <draft-ietf-asid-ldapv3-attributes-06.txt>, 11 July 1997.

[3] T. Howes, "A String Representation of LDAP Search Filters," INTERNET-DRAFT <draft-ietf-asid-ldapv3-filter-02.txt>, May 1997.

[4] S. Kille, M. Wahl, "A UTF-8 String Representation of Distinguished Names", INTERNET-DRAFT <draft-ietf-asid-ldapv3-dn-03.txt>, 29 April 1997.

[5] S. Kille, "Using the OSI Directory to Achieve User Friendly Naming," RFC 1781, March 1995.

[6] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", INTERNET-DRAFT <draft-ietf-asid-ldapv3-protocol-06.txt>, 11 July 1997.

[7] A. Herron, T. Howes, M. Wahl, "LDAP Control Extension for Server Side Sorting of Search Result," INTERNET-DRAFT <draft-ietf-asid-ldapv3-sorting-00.txt>, 16 April 1997.

[8] J. Meyers, "Simple Authentication and Security Layer", INTERNET-DRAFT <draft-myers-auth-sasl-11.txt>, April 1997.

[9] "Lightweight Directory Access Protocol (v3) Extension for Transport Layer Security", INTERNET-DRAFT <draft-ietf-asid-ldapv3-tls-01.txt>, June 1997.

[10] "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

[11] "IP Version 6 Addressing Architecture,", RFC 1884, December 1995.

17. Author's Addresses

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd., Mailstop MV068
Mountain View, CA 94043
USA
+1 415 937-3419
howes@netscape.com

Mark Smith
Netscape Communications Corp.
501 E. Middlefield Rd., Mailstop MV068
Mountain View, CA 94043
USA
+1 415 937-3477
mcs@netscape.com

Andy Herron
Microsoft Corp.
1 Microsoft Way
Redmond, WA 98052
USA
+1 425 882-8080
andyhe@microsoft.com

Chris Weider
Microsoft Corp.
1 Microsoft Way
Redmond, WA 98052
USA
+1 425 882-8080
cweider@microsoft.com

Mark Wahl
Critical Angle Inc.
4815 W Braker Lane #502-385
Austin, TX 78759
USA
M.Wahl@critical-angle.com

18. Appendix A - Sample LDAP API Code

```
#include <ldap.h>

main()
```

```
{
        LDAP            *ld;
        LDAPMessage     *res, *e;
        int             i;
        char            *a, *dn;
        BerElement      *ptr;
        char            **vals;

        /* open an LDAP session */
        if ( (ld = ldap_init( "dotted.host.name", LDAP_PORT )) == NULL )
                exit( 1 );

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

        /* search for entries with cn of "Babs Jensen", return all attrs  */
        if ( ldap_search_s( ld, "o=University of Michigan, c=US",
            LDAP_SCOPE_SUBTREE, "(cn=Babs Jensen)", NULL, 0, &res )
            != LDAP_SUCCESS ) {
                ldap_perror( ld, "ldap_search_s" );
                exit( 1 );
        }

        /* step through each entry returned */
        for ( e = ldap_first_entry( ld, res ); e != NULL;
            e = ldap_next_entry( ld, e ) ) {
                /* print its name */
                dn = ldap_get_dn( ld, e );
                printf( "dn: %s\n", dn );
                ldap_memfree( dn );

                /* print each attribute */
                for ( a = ldap_first_attribute( ld, e, &ptr ); a != NULL;
                    a = ldap_next_attribute( ld, e, ptr ) ) {
                        printf( "attribute: %s\n", a );

                        /* print each value */
                        vals = ldap_get_values( ld, e, a );
                        for ( i = 0; vals[i] != NULL; i++ ) {
                                printf( "value: %s\n", vals[i] );
                        }
                        ldap_value_free( vals );
                }
                if ( ptr != NULL ) {
                        ldap_ber_free( ptr, 0 );
```

```
                }
        }
        /* free the search results */
        ldap_msgfree( res );

        /* close and free connection resources */
        ldap_unbind( ld );
}
```

19. Appendix B - Outstanding Issues

19.1. Support for multithreaded applications

In order to support multithreaded applications in a platform-independent way, some additions to the LDAP API are needed. Different implementors have taken different paths to solve this problem in the past. A common set of thread-related API calls must be defined so that application developers are not unduly burdened. These will be added to a future revision of this specification.

19.2. Using Transport Layer Security (TLS)

The API calls used to support TLS must be specified. They will be added to a future revision of this specification.

19.3. Client control for chasing referrals

A client control has been defined that can be used to specify on a per-operation basis whether references and external referrals are automatically chased by the client library. This will be added to a future revision of this specification.

19.4. Potential confusion between hostname:port and IPv6 addresses

String representations of IPv6 network addresses [11] can contain colon characters. The ldap_init() call is specified to take strings of the form "hostname:port" or "ipaddress:port". If IPv6 addresses are used, the latter could be ambiguous. A future revision of this specification will resolve this issue.

19.5. Need to track SASL API standardization efforts

If a standard Simple Authentication and Security Layer API is defined, it may be necessary to modify the LDAP API to accommodate it.

19.6. Support for character sets other than UTF-8?

Some application developers would prefer to pass string data using a character set other than UTF-8. This could be accommodated by adding a new option to ldap_set_option() that supports choosing a character set. If this feature is added, the number of different character sets supported should definitely be minimized.

19.7. Use of UTF-8 with LDAPv2 servers

Strings are always passed as UTF-8 in this API but LDAP version 2 servers do not support the full range of UTF-8 characters. The expected behavior of this API when using LDAP version 2 with unsupported characters should be specified.