

## The LDAP Application Program Interface

### Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

### 1. Introduction

This document defines a C language application program interface to the lightweight directory access protocol (LDAP). The 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.

### 2. Overview of the LDAP Model

LDAP is the lightweight directory access protocol, described in [2] and [7]. 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 are 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 root of the tree down to the entry. 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.

### 3. Overview of LDAP API Use

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

- o Open a connection to an LDAP server. The `ldap_open()` call returns a handle to the connection, allowing multiple connections to be open at once.
- o Authenticate to the LDAP server and/or the X.500 DSA. The `ldap_bind()` call and friends support a variety of authentication methods.
- o 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.
- o Close the connection. The `ldap_unbind()` call closes the connection.

Operations can be performed either synchronously or asynchronously. Synchronous calls 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. The next sections describe these routines in more detail.

#### 4. Calls for performing LDAP operations

This section describes each LDAP operation API call in detail. All calls take a "connection 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.

##### 4.1. Opening a connection

ldap\_open() opens a connection to the LDAP server.

```
typedef struct ldap {
    /* ... opaque parameters ... */
    int     ld_deref;
    int     ld_timelimit;
    int     ld_sizelimit;
    int     ld_errno;
    char    *ld_matched;
    char    *ld_error;
    /* ... opaque parameters ... */
} LDAP;

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. The hosts are tried in the order listed, stopping with the first one to which a successful connection is made;

portno contains the TCP port number to which to connect. The default LDAP port can be obtained by supplying the constant LDAP\_PORT.

ldap\_open() returns a "connection handle", a pointer to an LDAP structure that should be passed to subsequent calls pertaining to the connection. It returns NULL if the connection cannot be opened. One of the ldap\_bind calls described below must be completed before other operations can be performed on the connection.

The calling program should assume nothing about the order of the fields in the LDAP structure. There may be other fields in the structure for internal library use. The fields shown above are described as needed in the description of other calls below.

#### 4.2. Authenticating to the directory

`ldap_bind()` and friends are used to authenticate to the directory.

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

Parameters are:

`ld` The connection handle;

`dn` The name of the entry to bind as;

`cred` The credentials with which to authenticate;

`method` One of `LDAP_AUTH_SIMPLE`, `LDAP_AUTH_KRBV41`, or `LDAP_AUTH_KRBV42`, indicating the authentication method to use;

`passwd` For `ldap_simple_bind()`, the password to compare to the entry's `userPassword` attribute;

There are three types of bind calls, providing simple authentication, kerberos authentication, and general routines to do either one. In the case of Kerberos version 4 authentication using the general `ldap_bind()` routines, the credentials are ignored, as the routines assume a valid ticket granting ticket already exists which can be used to retrieve the appropriate service tickets.

Synchronous versions of the routines have names that end in `_s`. These routines return the result of the bind 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.

Asynchronous versions of these routines return the message id of the bind operation initiated. A subsequent call to `ldap_result()`, described below, can be used to obtain the result of the bind. In case of error, these routines will return -1, setting the `ld_errno` field in the LDAP structure appropriately.

Note that 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.

#### 4.3. Closing the connection

`ldap_unbind()` is used to unbind from the directory and close the connection.

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

Parameters are:

`ld` The connection handle.

`ldap_unbind()` works synchronously, unbinding from the directory, closing the connection, and freeing up the `ld` structure before returning. `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()`, the `ld` connection handle is invalid.

#### 4.4. Searching

`ldap_search()` and friends are used to search the LDAP directory, returning a requested set of attributes for each entry matched. There are three variations.

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

**base**         The dn of the entry at which to start the search;

**scope**        One of LDAP\_SCOPE\_BASE, LDAP\_SCOPE\_ONELEVEL, or LDAP\_SCOPE\_SUBTREE, indicating the scope of the search;

**filter**       A character string as described in RFC 1558 [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() call, this specifies the local search timeout value;

**res**          For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call.

There are three fields in the ld connection handle which control how the search is performed. They are:

`ld_sizelimit` A limit on the number of entries to return from the search. A value of zero means no limit;

`ld_timelimit` A limit on the number of seconds to spend on the search. A value of zero means no limit;

`ld_deref` One of `LDAP_DEREF_NEVER`, `LDAP_DEREF_SEARCHING`, `LDAP_DEREF_FINDING`, or `LDAP_DEREF_ALWAYS`, 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.

An asynchronous search is initiated by calling `ldap_search()`. It returns the message id of the initiated search. The results of the search can be obtained by a subsequent call to `ldap_result()`. The results can be parsed by the result parsing routines described in detail later. In case of error, -1 is returned and the `ld_errno` field in the LDAP structure is set appropriately.

A synchronous search is performed by calling `ldap_search_s()` or `ldap_search_st()`. The routines are identical, except that `ldap_search_st()` takes an additional parameter specifying a timeout for the search. Both routines return an indication of the result of the search, either `LDAP_SUCCESS` or some error indication (see Error Handling below). The 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.

#### 4.5. 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.

#### 4.6. 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.

#### 4.7. Modifying an entry

The `ldap_modify()` and `ldap_modify_s()` routines are used to modify an existing LDAP entry.

```
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, char *dn, LDAPMod *mods[] );

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

Parameters are:

`ld`        The connection handle;

`dn`        The name of the entry to modify;

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

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`, `LDAP_MOD_DELETE`, or `LDAP_MOD_REPLACE`. This field also indicates the type of values included in the `mod_vals` union. It is ORed with `LDAP_MOD_BVALUES` 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. All modifications are performed in the order in which they are listed.

ldap\_modify\_s() returns the LDAP error code resulting from the modify operation. This code can be interpreted by ldap\_perror() and friends.

ldap\_modify() returns the message id of the request it initiates, or -1 on error. The result of the operation can be obtained by calling ldap\_result().

#### 4.8. Modifying the RDN of an entry

The ldap\_modrdn() and ldap\_modrdn\_s() routines are used to change the name of an LDAP entry.

```
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 connection handle;
dn	The name of the entry whose RDN is to be changed;
newrdn	The new RDN to give the entry;
deleteoldrdn	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.

The `ldap_modrdn_s()` routine is synchronous, returning the LDAP error code indicating the outcome of the operation.

The `ldap_modrdn()` routine is asynchronous, returning the message id of the operation it initiates, or -1 in case of trouble. The result of the operation can be obtained by calling `ldap_result()`.

#### 4.9. Adding an entry

`ldap_add()` and `ldap_add_s()` are used to add entries to the LDAP directory.

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

Parameters are:

`ld` The connection 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.

Note that the parent of the entry must already exist.

`ldap_add_s()` is synchronous, returning the LDAP error code indicating the outcome of the operation.

`ldap_add()` is asynchronous, returning the message id of the operation it initiates, or -1 in case of trouble. The result of the operation can be obtained by calling `ldap_result()`.

#### 4.10. Deleting an entry

`ldap_delete()` and `ldap_delete_s()` are used to delete entries from the LDAP directory.

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

Parameters are:

ld           The connection handle;  
dn           The name of the entry to delete.

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

ldap\_delete\_s() is synchronous, returning the LDAP error code indicating the outcome of the operation.

ldap\_delete() is asynchronous, returning the message id of the operation it initiates, or -1 in case of trouble. The result of the operation can be obtained by calling ldap\_result().

#### 5. Calls for abandoning an operation

ldap\_abandon() is used to abandon an operation in progress.

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

ldap\_abandon() abandons the operation with message id msgid. It returns zero if the abandon was successful, -1 otherwise. After a successful call to ldap\_abandon(), results with the given message id are never returned from a call to ldap\_result().

#### 6. Calls for obtaining results

ldap\_result() is used to obtain the result of a previous asynchronously initiated operation. ldap\_msgfree() frees the results obtained from a previous call to ldap\_result(), or a synchronous search routine.

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

int ldap_msgfree( LDAPMessage *res );
```

Parameters are:

- `ld`           The connection handle;
- `msgid`        The message id of the operation whose results are to be returned, or the constant `LDAP_RES_ANY` if any result is desired;
- `all`           A boolean parameter that only has meaning for search results. If non-zero it indicates that all results of a search should be retrieved before any are returned. If zero, search results (entries) will be returned one at a time as they arrive;
- `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 second 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()` or `ldap_search_s()` or `ldap_search_st()`.

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

```
LDAP_RES_BIND
LDAP_RES_SEARCH_ENTRY
LDAP_RES_SEARCH_RESULT
LDAP_RES_MODIFY
LDAP_RES_ADD
LDAP_RES_DELETE
LDAP_RES_MODRDN
LDAP_RES_COMPARE
```

`ldap_result()` returns 0 if the timeout expired and -1 if an error occurs, in which case the `ld_errno` field of the `ld` structure will be set accordingly.

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

## 7. Calls for error handling

The following calls are used to interpret errors returned by other LDAP API routines.

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

char *ldap_err2string( int err );

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

Parameters are:

`ld`        The connection handle;

`res`        The result of an LDAP operation as returned by `ldap_result()` or one of the synchronous API operation calls;

`freeit`    A boolean parameter indicating whether the `res` parameter should be freed (non-zero) or not (zero);

`err`        An LDAP error code, as returned by `ldap_result2error()` or one of the synchronous API operation calls;

`msg`        A message to be displayed before the LDAP error message.

`ldap_result2error()` is used to convert the LDAP result message obtained from `ldap_result()`, or the `res` parameter returned by one of the synchronous API operation calls, into a numeric LDAP error code. It also parses the `ld_matched` and `ld_error` portions of the result message and puts them into the connection handle information. All the synchronous operation routines call `ldap_result2error()` before returning, ensuring that these fields are set correctly. The relevant fields in the connection structure are:

`ld_matched` In the event of an `LDAP_NO_SUCH_OBJECT` error return, this parameter contains the extent of the DN matched;

`ld_error`    This parameter contains the error message sent in the result by the LDAP server.

`ld_errno`    The LDAP error code indicating the outcome of the operation. It is one of the following constants:

LDAP\_SUCCESS  
LDAP\_OPERATIONS\_ERROR  
LDAP\_PROTOCOL\_ERROR  
LDAP\_TIMELIMIT\_EXCEEDED  
LDAP\_SIZELIMIT\_EXCEEDED  
LDAP\_COMPARE\_FALSE  
LDAP\_COMPARE\_TRUE  
LDAP\_STRONG\_AUTH\_NOT\_SUPPORTED  
LDAP\_STRONG\_AUTH\_REQUIRED  
LDAP\_NO\_SUCH\_ATTRIBUTE  
LDAP\_UNDEFINED\_TYPE  
LDAP\_INAPPROPRIATE\_MATCHING  
LDAP\_CONSTRAINT\_VIOLATION  
LDAP\_TYPE\_OR\_VALUE\_EXISTS  
LDAP\_INVALID\_SYNTAX  
LDAP\_NO\_SUCH\_OBJECT  
LDAP\_ALIAS\_PROBLEM  
LDAP\_INVALID\_DN\_SYNTAX  
LDAP\_IS\_LEAF  
LDAP\_ALIAS\_DEREF\_PROBLEM  
LDAP\_INAPPROPRIATE\_AUTH  
LDAP\_INVALID\_CREDENTIALS  
LDAP\_INSUFFICIENT\_ACCESS  
LDAP\_BUSY  
LDAP\_UNAVAILABLE  
LDAP\_UNWILLING\_TO\_PERFORM  
LDAP\_LOOP\_DETECT  
LDAP\_NAMING\_VIOLATION  
LDAP\_OBJECT\_CLASS\_VIOLATION  
LDAP\_NOT\_ALLOWED\_ON\_NONLEAF  
LDAP\_NOT\_ALLOWED\_ON\_RDN  
LDAP\_ALREADY\_EXISTS  
LDAP\_NO\_OBJECT\_CLASS\_MODS  
LDAP\_RESULTS\_TOO\_LARGE  
LDAP\_OTHER  
LDAP\_SERVER\_DOWN  
LDAP\_LOCAL\_ERROR  
LDAP\_ENCODING\_ERROR  
LDAP\_DECODING\_ERROR  
LDAP\_TIMEOUT  
LDAP\_AUTH\_UNKNOWN  
LDAP\_FILTER\_ERROR  
LDAP\_USER\_CANCELLED  
LDAP\_PARAM\_ERROR  
LDAP\_NO\_MEMORY

`ldap_err2string()` is used to convert a numeric LDAP error code, as returned by `ldap_result2error()` 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.

`ldap_perror()` is used to print the message supplied in `msg`, followed by an indication of the error contained in the `ld_errno` field of the `ld` connection handle, to standard error.

## 8. Calls for parsing search entries

The following calls are used to parse the entries returned by `ldap_search()` and friends. These entries 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 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.

### 8.1. Stepping through a set of entries

The `ldap_first_entry()` and `ldap_next_entry()` routines are used to step through a set of entries in a search result. `ldap_count_entries()` is used to count the number of entries returned.

```
LDAPMessage *ldap_first_entry( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_entry( LDAP *ld, LDAPMessage *entry );
int ldap_count_entries( LDAP *ld, LDAPMessage *res );
```

Parameters are:

`ld`      The connection 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 exist to be returned. NULL is also returned if an error occurs while stepping through the entries, in which case the `ld_errno` field of the `ld` connection handle 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 an entry returned by `ldap_first_entry()` or `ldap_next_entry()`.

## 8.2. 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,
    void          **ptr
);
char *ldap_next_attribute(
    LDAP          *ld,
    LDAPMessage   *entry,
    void          *ptr
);
```

Parameters are:

`ld`      The connection 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()`.

`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 `ld_errno` field in the `ld` connection handle will be set to indicate the error.

Both routines return a pointer to a per-connection buffer containing the current attribute name. This should be treated like static data. `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.

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

### 8.3. 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.

```

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

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 connection 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 only. The second `_len` form is used with any kind of data.

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

#### 8.4. Retrieving the name of an entry

`ldap_get_dn()` is used to retrieve the name of an entry. `ldap_explode_dn()` is used to break up the 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_dn2ufn( char *dn );
```

Parameters are:

`ld`        The connection 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, as returned by `ldap_get_dn()`;

`notypes`   A boolean parameter, if non-zero indicating that the dn 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 `ld_errno` in the ld connection handle to indicate the error. It returns a pointer to malloc'ed space that the caller should free by calling `free()` when it is no longer in use. Note the format of the DNs returned is given by [4].

`ldap_explode_dn()` returns a 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_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 `free()` when no longer in use.

## 9. Security Considerations

LDAP supports minimal security during connection authentication.

## 10. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. NCR-9416667.

## 11. Bibliography

- [1] The Directory: Selected Attribute Syntaxes. CCITT, Recommendation X.520.
- [2] Howes, T., Kille, S., Yeong, W., and C. Robbins, "The String Representation of Standard Attribute Syntaxes", University of Michigan, ISODE Consortium, Performance Systems International, NeXor Ltd., RFC 1778, March 1995.
- [3] Howes, T., "A String Representation of LDAP Search Filters", RFC 1558, University of Michigan, December 1993.
- [4] Kille, S., "A String Representation of Distinguished Names", RFC 1779, ISODE Consortium, March 1995.
- [5] Kille, S., "Using the OSI Directory to Achieve User Friendly Naming", RFC 1781, ISODE Consortium, March 1995.
- [6] S.P. Miller, B.C. Neuman, J.I. Schiller, J.H. Saltzer, "Kerberos Authentication and Authorization System", MIT Project Athena Documentation Section E.2.1, December 1987
- [7] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol," RFC 1777, Performance Systems International, University of Michigan, ISODE Consortium, March 1995.

## 12. Authors' Addresses

Tim Howes  
University of Michigan  
ITD Research Systems  
535 W William St.  
Ann Arbor, MI 48103-4943  
USA

Phone: +1 313 747-4454  
EMail: tim@umich.edu

Mark Smith  
University of Michigan  
ITD Research Systems  
535 W William St.  
Ann Arbor, MI 48103-4943  
USA

Phone: +1 313 764-2277  
EMail: mcs@umich.edu

## 13. Appendix A - Sample LDAP API Code

```

#include <ldap.h>

main()
{
    LDAP          *ld;
    LDAPMessage   *res, *e;
    int           i;
    char          *a, *dn;
    void          *ptr;
    char          **vals;

    /* open a connection */
    if ( (ld = ldap_open( "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: %s0, dn );
        free( dn );

        /* print each attribute */
        for ( a = ldap_first_attribute( ld, e, &ptr );
              a != NULL;
              a = ldap_next_attribute( ld, e, ptr ) ) {
            printf( "attribute: %s0, a );

            /* print each value */

```

```
        vals = ldap_get_values( ld, e, a );
        for ( i = 0; vals[i] != NULL; i++ ) {
            printf( "value: %s0, vals[i] );
        }
        ldap_value_free( vals );
    }
}
/* free the search results */
ldap_msgfree( res );

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

