Network Working Group
INTERNET-DRAFT
Intended Category: Standards Track
Obsoletes: RFC 1823
Expires: May 2001

M. Smith, Editor
Netscape Communications Corp.
T. Howes
Loudcloud, Inc.
A. Herron
Microsoft Corp.
M. Wahl
Sun Microsystems, Inc.
A. Anantha
Microsoft Corp.

17 November 2000

The C LDAP Application Program Interface
<draft-ietf-ldapext-ldap-c-api-05.txt>

1. Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026. 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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This draft document will be submitted to the RFC Editor as a Standards Track document. Distribution of this memo is unlimited. Technical discussion of this document will take place on the IETF LDAP Extension Working Group mailing list <ietf-ldapext@netscape.com>. Please send editorial comments directly to the authors.

2. Introduction

This document defines a C language application program interface (API) 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 key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119[1].

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 appendices that provide example code demonstrating use of the API, the namespace consumed by the API, a summary of requirements for API extensions, known incompatibilities with RFC 1823, and a list of changes made since the last revision of this document.

3. Table of Contents

4. Overview of the LDAP Model

LDAP is the lightweight directory access protocol, described in [2] and [3]. It can provide a lightweight frontend to the X.500 directory [4], 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 [5].

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.

5. Overview of LDAP API Use and General Requirements

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

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

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

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

4. 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 result code). The asynchronous routines make available to the caller 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() or ldap_abandon_ext(). Note that there is no requirement that an LDAP API implementation not block when handling asynchronous API functions; the term "asynchronous" as used in this document refers to the fact that the sending of LDAP requests can be separated from the receiving of LDAP responses.

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 can return referrals and references 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.

All DN and string attribute values passed into or produced by this C LDAP API are represented using the character set of the underlying LDAP protocol version in use. When this API is used with LDAPv3, DN and string values are represented as UTF-8[6] characters. When this API is used with LDAPv2, the US-ASCII[7] or T.61[7] character set are used. Future documents MAY specify additional APIs supporting other character sets.

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.

Unless otherwise indicated, conformant implementations of this specification MUST implement all of the C LDAP API functions as described in this document, and they MUST use the function prototypes, macro definitions, and types defined in this document.

Note that this API is designed for use in environments where the 'int' type is at least 32 bits in size.

6. Header Requirements

To promote portability of applications, the following requirements are imposed on the headers used by applications to access the services of this API:

Name and Inclusion

Applications only need to include a single header named ldap.h to access all of the API services described in this document. Therefore, the following C source program MUST compile and execute without errors:

```
#include <ldap.h>

int
main()
{
    return 0;
}
```

The ldap.h header MAY include other implementation-specific headers.

Implementations SHOULD also provide a header named lber.h to facilitate development of applications desiring compatibility with older LDAP implementations. The lber.h header MAY be empty. Old applications that include lber.h in order to use BER facilities will need to include ldap.h.

Idempotence

All headers SHOULD be idempotent; that is, if they are included more than once the effect is as if they had only been included

once.

Must Be Included Before API Is Used
: An application MUST include the ldap.h header before referencing any of the function or type definitions described in this API specification.

Mutual Independence
: Headers SHOULD be mutually independent with minimal dependence on system or any other headers.

Use of the 'const' Keyword
: This API specification is defined in terms of ISO C[8]. It makes use of function prototypes and the 'const' keyword. The use of 'const' in this specification is limited to simple, non-array function parameters to avoid forcing applications to declare parameters and variables that accept return values from LDAP API functions as 'const.' Implementations specifically designed to be used with non-ISO C translators SHOULD provide function declarations without prototypes or function prototypes without specification of 'const' arguments.

Definition of 'struct timeval'
: This API specification uses the 'struct timeval' type. Implementations of this API MUST ensure that the struct timeval type is by default defined as a consequence of including the ldap.h header. Because struct timeval is usually defined in one or more system headers, it is possible for header conflicts to occur if ldap.h also defines it or arranges for it to be defined by including another header. Therefore, applications MAY want to arrange for struct timeval to be defined before they include ldap.h. To support this, the ldap.h header MUST NOT itself define struct timeval if the preprocessor symbol LDAP_TYPE_TIMEVAL_DEFINED is defined before ldap.h is included.

7. Common Data Structures and Types

Data structures and types that are common to several LDAP API functions are defined here:

```
typedef struct ldap LDAP;

typedef struct ldapmsg LDAPMessage;

typedef struct berelement BerElement;

typedef <impl_len_t> ber_len_t;
```

```
typedef struct berval {
    ber_len_t       bv_len;
    char            *bv_val;
} BerValue;

struct timeval {
    <impl_sec_t>    tv_sec;
    <impl_usec_t>   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 entry, reference, result, and error information. An LDAPMessage structure can represent the beginning of a list, or chain of messages that consists of a series of entries, references, and result messages as returned by LDAP operations such as search. LDAP API functions such as ldap_parse_result() that operate on message chains that can contain more than one result message always operate on the first result message in the chain. See the "Obtaining Results and Peeking Inside LDAP Messages" section of this document for more information.

The BerElement structure is an opaque data type that is used to hold data and state information about encoded data. It is described in more detail in the section "Encoded ASN.1 Value Manipulation" later in this document.

The `ber_len_t' type is an unsigned integral data type that is large enough to contain the length of the largest piece of data supported by the API implementation. The `<impl_len_t>' in the `ber_len_t' typedef MUST be replaced with an appropriate type. The width (number of significant bits) of `ber_len_t' MUST be at least 32 and no larger than that of `unsigned long'. See the appendix "Data Types and Legacy Implementations" for additional considerations.

The BerValue 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.

Note that because the struct timeval definition typically is derived from a system header, the types used for the tv_sec and tv_usec components are implementation-specific integral types. Therefore, `<impl_sec_t>' and `<impl_usec_t>' in the struct timeval definition MUST be replaced with appropriate types. See the earlier section "Header Requirements" for more information on struct timeval.

8. Memory Handling Overview

All memory that is allocated by a function in this C LDAP API and returned to the caller SHOULD be disposed of by calling the appropriate "free" function provided by this API. The correct "free" function to call is documented in each section of this document where a function that allocates memory is described.

Memory that is allocated through means outside of the C LDAP API MUST NOT be disposed of using a function provided by this API.

If a pointer value passed to one of the C LDAP API "free" functions is NULL, graceful failure (i.e, ignoring of the NULL pointer) MUST occur.

The complete list of "free" functions that are used to dispose of allocated memory is:

```
ber_bvecfree()
ber_bvfree()
ber_free()
ldap_control_free()
ldap_controls_free()
ldap_memfree()
ldap_msgfree()
ldap_value_free()
ldap_value_free_len()
```

9. Retrieving Information About the API Implementation

Applications developed to this specification need to be able to determine information about the particular API implementation they are using both at compile time and during execution.

9.1. Retrieving Information at Compile Time

All conformant implementations MUST include the following five definitions in a header so compile time tests can be done by LDAP software developers:

```
#define LDAP_API_VERSION     level
#define LDAP_VERSION_MIN     min-version
#define LDAP_VERSION_MAX     max-version
#define LDAP_VENDOR_NAME     "vend-name"
#define LDAP_VENDOR_VERSION  vend-version
```

where:

"level" is replaced with the RFC number given to this C LDAP API specification when it is published as a standards track RFC.

min-version is replaced with the lowest LDAP protocol version supported by the implementation.

max-version is replaced with the highest LDAP protocol version supported by the implementation. This SHOULD be 3.

"vend-name" is replaced with a text string that identifies the party that supplies the API implementation.

"vend-version" is a supplier-specific version number multiplied times 100.

Note that the LDAP_VENDOR_NAME macro SHOULD be defined as "" if no vendor name is available and the LDAP_VENDOR_VERSION macro SHOULD be defined as 0 if no vendor-specific version information is available.

For example, if this specification is published as RFC 88888, Netscape Communication's version 4.0 implementation that supports LDAPv2 and v3 might include macro definitions like these:

```
#define LDAP_API_VERSION     88888      /* RFC 88888 compliant */
#define LDAP_VERSION_MIN     2
#define LDAP_VERSION_MAX     3
#define LDAP_VENDOR_NAME     "Netscape Communications Corp."
#define LDAP_VENDOR_VERSION  400        /* version 4.0 */
```

and application code can test the C LDAP API version level using a construct such as this one:

```
#if (LDAP_API_VERSION >= 88888)
        /* use features supported in RFC 88888 or later */
```

```
#endif
```

Until such time as this document is published as an RFC, implementations SHOULD use the value 2000 plus the revision number of this draft for LDAP_API_VERSION. For example, the correct value for LDAP_API_VERSION for revision 05 of this draft is 2005.

Documents that extend this specification SHOULD define a macro of the form:

```
#define LDAP_API_FEATURE_x level
```

where "x" is replaced with a name (textual identifier) for the feature and "level" is replaced with the number of the RFC that specifies the API extension. The name SHOULD NOT begin with the string "X_".

For example, if C LDAP API extensions for Transport Layer Security [9] were published in RFC 99999, that RFC might require conformant implementations to define a macro like this:

```
#define LDAP_API_FEATURE_TLS 99999
```

Private or experimental API extensions SHOULD be indicated by defining a macro of this same form where "x" (the extension's name) begins with the string "X_" and "level" is replaced with a integer number that is specific to the extension.

It is RECOMMENDED that private or experimental API extensions use only the following prefixes for macros, types, and function names:

- LDAP_X_
- LBER_X_
- ldap_x_
- ber_x_

and that these prefixes not be used by standard extensions.

9.2. Retrieving Information During Execution

The ldap_get_option() call (described in greater detail later in this document) can be used during execution in conjunction with an option parameter value of LDAP_OPT_API_INFO (0x00) to retrieve some basic information about the API and about the specific implementation being used. The ld parameter to ldap_get_option() can be either NULL or a valid LDAP session handle which was obtained by calling ldap_init(). The optdata parameter to ldap_get_option() SHOULD be the address of an LDAPAPIInfo structure which is defined as follows:

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

In addition, API implementations MUST include the following macro definition:

```
   #define LDAP_API_INFO_VERSION    1
```

Note that the ldapai_info_version field of the LDAPAPIInfo structure SHOULD be set to the value LDAP_API_INFO_VERSION (1) before calling ldap_get_option() so that it can be checked for consistency. All other fields are set by the ldap_get_option() function.

The members of the LDAPAPIInfo structure are:

ldapai_info_version
: A number that identifies the version of the LDAPAPIInfo structure. This SHOULD be set to the value LDAP_API_INFO_VERSION (1) before calling ldap_get_option(). If the value received is not recognized by the API implementation, the ldap_get_option() function sets ldapai_info_version to a valid value that would be recognized, sets the ldapai_api_version to the correct value, and returns an error without filling in any of the other fields in the LDAPAPIInfo structure.

ldapai_api_version
: A number that matches that assigned to the C LDAP API RFC supported by the API implementation. This SHOULD match the value of the LDAP_API_VERSION macro defined earlier.

ldapai_protocol_version
: The highest LDAP protocol version supported by the implementation. For example, if LDAPv3 is the highest version supported then this field will be set to 3.

ldapai_vendor_name
: A zero-terminated string that contains the name of the party that produced the LDAP API implementation. This field may be set to NULL if no name is available. If non-NULL, the caller is responsible for disposing of the memory occupied by passing this pointer to ldap_memfree() which is described later in this document. This value SHOULD match the value of the

LDAP_VENDOR_NAME macro described earlier in this document.

ldapai_vendor_version
An implementation-specific version number multiplied by 100. For example, if the implementation version is 4.0 then this field will be set to 400. If no version information is available, this field will be set to 0. This value SHOULD match the value of the LDAP_VENDOR_VERSION macro described earlier in this document.

ldapai_extensions
A NULL-terminated array of character strings that lists the names of the API extensions supported by the LDAP API implementation. These names will typically match the textual identifiers that appear in the "x" portion of the LDAP_API_FEATURE_x macros described above, although the precise value MUST be defined by documents that specify C LDAP API extensions. If no API extensions are supported, this field will be set to NULL. The caller is responsible for disposing of the memory occupied by this array by passing it to ldap_value_free() which is described later in this document. To retrieve more information about a particular extension, the ldap_get_option() call can be used with an option parameter value of LDAP_OPT_API_FEATURE_INFO (0x15). The optdata parameter to the ldap_get_option() SHOULD be the address of an LDAPAPIFeatureInfo structure which is defined as follows:

```
typedef struct ldap_apifeature_info {
    int   ldapaif_info_version; /* version of this struct (1) */
    char  *ldapaif_name;        /* name of supported feature */
    int   ldapaif_version;      /* revision of supported feature
*/
} LDAPAPIFeatureInfo;
```

In addition, API implementations MUST include the following macro definition:

```
#define LDAP_FEATURE_INFO_VERSION    1
```

Note that the ldapaif_info_version field of the LDAPAPIFeatureInfo structure SHOULD be set to the value LDAP_FEATURE_INFO_VERSION (1) and the ldapaif_name field SHOULD be set to the extension name string as described below before ldap_get_option() is called. The call will fill in the ldapaif_version field of the LDAPAPIFeatureInfo structure.

The members of the LDAPAPIFeatureInfo structure are:

ldapaif_info_version
: A number that identifies the version of the LDAPAPI-FeatureInfo structure. This SHOULD be set to the value LDAP_FEATURE_INFO_VERSION (1) before calling ldap_get_option(). If the value received is not recognized by the API implementation, the ldap_get_option() function sets ldapaif_info_version to a valid value that would be recognized and returns an error without filling in the ldapaif_version field in the LDAPAPIFeatureInfo structure.

ldapaif_name
: The name of an extension, as returned in the ldapai_extensions array of the LDAPAPIInfo structure and as specified in the document that describes the extension.

ldapaif_version
: This field will be set as a result of calling ldap_get_option(). It is a number that matches that assigned to the C LDAP API extension RFC supported for this extension. For private or experimental API extensions, the value is extension-specific. In either case, the value of ldapaxi_ext_version SHOULD be identical to the value of the LDAP_API_FEATURE_x macro defined for the extension (described above).

10. Result Codes

Many of the LDAP API routines return result codes, some of which indicate local API errors and some of which are LDAP resultCodes that are returned by servers. All of the result codes are non-negative integers. Supported result codes are as follows (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_SASL_BIND_IN_PROGRESS (0x0e)             -- 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)                -- reserved for
CLDAP
            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)
```

11. Performing LDAP Operations

This section describes each LDAP operation API call in detail. Most 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.

11.1. 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(
        const char      *hostname,
        int             portno
);
```

Use of the following routine is deprecated:

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

Unlike ldap_init(), ldap_open() attempts to make a server connection before returning to the caller. A more complete description can be found in RFC 1823.

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 MAY include a port number which is separated from the host itself with a colon (:) character. The hosts will be tried in the order listed, stopping with the first one to which a successful connection is made.

Note: A suitable representation for including a literal IPv6[10] address in the hostname parameter is desired, but has not yet been determined or implemented in practice.

portno Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the value zero (0) or the macro LDAP_PORT (389). 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 MUST 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 SHOULD be completed before other operations can be performed on the session. LDAPv3 does not require that a bind operation be completed before other operations can be performed.

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

11.2. LDAP Session Handle Options

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

In the interest of insulating callers from inevitable changes to this structure, these aspects of the session are now accessed through a pair of accessor functions, described 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. Note that some options are READ-ONLY and cannot be set; it is an error to call ldap_set_option() and attempt to set a READ-ONLY option.

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

```
int ldap_get_option(
        LDAP            *ld,
        int             option,
        void            *outvalue
);

int ldap_set_option(
        LDAP            *ld,
        int             option,
        const void      *invalue
```

```
          );

          #define LDAP_OPT_ON     (<impl_void_star_value>)
          #define LDAP_OPT_OFF    ((void *)0)
```

LDAP_OPT_ON MUST be defined as a non-null pointer to void; that is, <impl_void_star_value> MUST be replaced with a non-null pointer to void, e.g., one could use:

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

if that value is safe to use on the architecture where the API is implemented.

Parameters are:

ld The session handle. If this is NULL, a set of global defaults is accessed. New LDAP session handles created with ldap_init() or ldap_open() inherit their characteristics from these global defaults.

option The name of the option being accessed or set. This parameter SHOULD be one of the following constants, which have the indicated meanings. After the constant the actual hexadecimal value of the constant is listed in parentheses.

LDAP_OPT_API_INFO (0x00)

Type for invalue parameter: not applicable (option is READ-ONLY)

Type for outvalue parameter: LDAPAPIInfo *

Description:

Used to retrieve some basic information about the LDAP API implementation at execution time. See the section "Retrieving Information About the API Implementation" above for more information. This option is READ-ONLY and cannot be set.

LDAP_OPT_DEREF (0x02)

Type for invalue parameter: int *

Type for outvalue parameter: int *

Description:

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

LDAP_DEREF_FINDING value means aliases are dereferenced when locating the base object but not during the search. The default value for this option is LDAP_DEREF_NEVER.

LDAP_OPT_SIZELIMIT (0x03)
Type for invalue parameter: int *

Type for outvalue parameter: int *

Description:
A limit on the number of entries to return from a search. A value of LDAP_NO_LIMIT (0) means no limit. The default value for this option is LDAP_NO_LIMIT.

LDAP_OPT_TIMELIMIT (0x04)
Type for invalue parameter: int *

Type for outvalue parameter: int *

Description:
A limit on the number of seconds to spend on a search. A value of LDAP_NO_LIMIT (0) means no limit. The default value for this option is LDAP_NO_LIMIT. This value is passed to the server in the search request only; it does not affect how long the C LDAP API implementation itself will wait locally for search results. Note that the timeout parameter passed to the ldap_search_ext_s() or ldap_result() functions can be used to specify a limit on how long the API implementation will wait for results.

LDAP_OPT_REFERRALS (0x08)
Type for invalue parameter: void * (LDAP_OPT_ON or LDAP_OPT_OFF)

Type for outvalue parameter: int *

Description:
Determines whether the LDAP library automatically follows referrals returned by LDAP servers or not. It MAY be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF; any non-NULL pointer value passed to ldap_set_option() enables this option. When reading the current setting using ldap_get_option(), a zero value means OFF and any non-zero value means ON. By default, this option is ON.

LDAP_OPT_RESTART (0x09)
Type for invalue parameter: void * (LDAP_OPT_ON or LDAP_OPT_OFF)

Type for outvalue parameter: int *

Description:
Determines whether LDAP I/O operations are automatically restarted if they abort prematurely. It MAY be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF; any non-NULL pointer value passed to ldap_set_option() enables this option. When reading the current setting using ldap_get_option(), a zero value means OFF and any non-zero value means ON. This option is useful if an LDAP I/O operation can be interrupted prematurely, for example by a timer going off, or other interrupt. By default, this option is OFF.

LDAP_OPT_PROTOCOL_VERSION (0x11)
Type for invalue parameter: int *

Type for outvalue parameter: int *

Description:
This option indicates the version of the LDAP protocol used when communicating with the primary LDAP server. It SHOULD be one of the constants LDAP_VERSION2 (2) or LDAP_VERSION3 (3). If no version is set the default is LDAP_VERSION2 (2).

LDAP_OPT_SERVER_CONTROLS (0x12)
Type for invalue parameter: LDAPControl **

Type for outvalue parameter: LDAPControl ***

Description:
A default list of LDAP server controls to be sent with each request. See the Working With Controls section below.

LDAP_OPT_CLIENT_CONTROLS (0x13)
Type for invalue parameter: LDAPControl **

Type for outvalue parameter: LDAPControl ***

Description:
A default list of client controls that affect the LDAP session. See the Working With Controls section below.

LDAP_OPT_API_FEATURE_INFO (0x15)
Type for invalue parameter: not applicable (option is READ-ONLY)

Type for outvalue parameter: LDAPAPIFeatureInfo *

Description:
Used to retrieve version information about LDAP API extended features at execution time. See the section "Retrieving

Information About the API Implementation" above for more information. This option is READ-ONLY and cannot be set.

LDAP_OPT_HOST_NAME (0x30)
Type for invalue parameter: char *

Type for outvalue parameter: char **

Description:
The host name (or list of hosts) for the primary LDAP server. See the definition of the hostname parameter to ldap_init() for the allowed syntax. Note that if the portno parameter passed to ldap_init() is a value other than 0 or 389 (LDAP_PORT), this value SHOULD include a string like ":portno" after each hostname or IP address that did not have one in the original hostname parameter that was passed to ldap_init(). For example, if this hostname value was passed to ldap_init():

"ldap.example.com:389 ldap2.example.com"

and the portno parameter passed to ldap_init() was 6389, then the value returned for the LDAP_OPT_HOST_NAME option SHOULD be:

"ldap.example.com:389 ldap2.example.com:6389"

LDAP_OPT_RESULT_CODE (0x31)
Type for invalue parameter: int *

Type for outvalue parameter: int *

Description:
The most recent local (API generated) or server returned LDAP result code that occurred for this session.

LDAP_OPT_ERROR_STRING (0x32)
Type for invalue parameter: char *

Type for outvalue parameter: char **

Description:
The message returned with the most recent LDAP error that occurred for this session.

LDAP_OPT_MATCHED_DN (0x33)
Type for invalue parameter: char *

Type for outvalue parameter: char **

Description:
The matched DN value 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. For outvalues of type char ** and LDAPControl **, a copy of the data that is associated with the LDAP session ld is returned; callers should dispose of the memory by calling ldap_memfree() or ldap_controls_free(), depending on the type of data returned.

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

Both ldap_get_option() and ldap_set_option() return 0 if successful and -1 if an error occurs. If -1 is returned by either function, a specific result code MAY be retrieved by calling ldap_get_option() with an option value of LDAP_OPT_RESULT_CODE. Note that there is no way to retrieve a more specific result code if a call to ldap_get_option() with an option value of LDAP_OPT_RESULT_CODE fails.

When a call to ldap_get_option() succeeds, the API implementation MUST NOT change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls. When a call to ldap_get_option() fails, the only session handle change permitted is setting the LDAP result code (as returned by the LDAP_OPT_RESULT_CODE option).

When a call to ldap_set_option() fails, it MUST NOT change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls.

Standards track documents that extend this specification and specify new options SHOULD use values for option macros that are between 0x1000 and 0x3FFF inclusive. Private and experimental extensions SHOULD use values for the option macros that are between 0x4000 and 0x7FFF inclusive. All values below 0x1000 and above 0x7FFF that are not defined in this document are reserved and SHOULD NOT be used. The following macro MUST be

defined by C LDAP API implementations to aid extension implementors:

```
#define LDAP_OPT_PRIVATE_EXTENSION_BASE 0x4000  /* to 0x7FFF inclusive */
```

11.3. Working With Controls

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

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

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

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). To specify a zero-length value, set ldctl_value.bv_len to zero and ldctl_value.bv_val to a zero-length string. To indicate that no data is associated with the control, set ldctl_value.bv_val to NULL.

ldctl_iscritical — Indicates whether the control is critical of not. If this field is non-zero, the operation will only be carried out if the control is recognized by the server and/or client. Note that the LDAP unbind and abandon operations have no server response, so clients SHOULD NOT mark server controls critical when used with these two operations.

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

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

If the ctrl or ctrls parameter is NULL, these calls do nothing.

A set of controls that affect the entire session can be set using the ldap_set_option() function (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 [11].

One client control is defined in this document (described in the following section). Other client controls MAY be defined in future revisions of this document or in documents that extend this API.

11.3.1. A Client Control That Governs Referral Processing

As described previously in the section "LDAP Session Handle Options," applications can enable and disable automatic chasing of referrals on a session-wide basic by using the ldap_set_option() function with the LDAP_OPT_REFERRALS option. It is also useful to govern automatic referral chasing on per-request basis. A client control with an OID of **1.2.840.113556.1.4.616 exists to provide this functionality.**

```
/* OID for referrals client control */
#define LDAP_CONTROL_REFERRALS              "1.2.840.113556.1.4.616"

/* Flags for referrals client control value */
#define LDAP_CHASE_SUBORDINATE_REFERRALS    0x00000020U
#define LDAP_CHASE_EXTERNAL_REFERRALS       0x00000040U
```

To create a referrals client control, the ldctl_oid field of an LDAPControl structure MUST be set to LDAP_CONTROL_REFERRALS ("1.2.840.113556.1.4.616") and the ldctl_value field MUST be set to a value that contains a set of flags. The ldctl_value.bv_len field MUST be set to sizeof(ber_uint_t), and the ldctl_value.bv_val field MUST point to a ber_uint_t which contains the flags value." The ber_uint_t type is define in the section "BER Data Structures and Types" below.

The flags value can be set to zero to disable automatic chasing of referrals and LDAPv3 references altogether. Alternatively, the flags value can be set to the value LDAP_CHASE_SUBORDINATE_REFERRALS (0x00000020U) to indicate that only LDAPv3 search continuation references are to be automatically chased by the API implementation, to the value LDAP_CHASE_EXTERNAL_REFERRALS (0x00000040U) to indicate that only LDAPv3 referrals are to be automatically chased, or the logical OR of the two flag values (0x00000060U) to indicate that both referrals and

references are to be automatically chased.

11.4. 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 [12]. 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 (NULL) 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,
        const char              *dn,
        const char              *mechanism,
        const struct berval     *cred,
        LDAPControl             **serverctrls,
        LDAPControl             **clientctrls,
        int                     *msgidp
);

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

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

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

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

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

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

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

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

Parameters are:

ld	The session handle.
dn	The name of the entry to bind as. If NULL, a zero length DN is sent to the server.
mechanism	Either LDAP_SASL_SIMPLE (NULL) to get simple authentication, or a text string identifying the SASL method.
cred	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter. If the cred parameter is NULL and the mechanism is LDAP_SASL_SIMPLE, a zero-length octet string is sent to the server in the simple credentials field of the bind request. If the cred parameter is NULL and the mechanism is anything else, no credentials are sent to the server in the bind request.
passwd	For ldap_simple_bind(), the password that is sent to the server in the simple credentials field of the bind request. If NULL, a zero length password is sent to the server.
serverctrls	List of LDAP server controls, or NULL if no server controls are used.
clientctrls	List of client controls, or NULL if no client controls are used.
msgidp	This result parameter will be set to the message id of the request if the ldap_sasl_bind() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

servercredp 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 ber_bvfree(). NULL SHOULD be passed to ignore this field. If an API error occurs or the server did not return any credentials, *servercredp is 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 result 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 result 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 can be attempted before a bind call has successfully completed.

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

11.5. Closing the session

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

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

```
int ldap_unbind( LDAP *ld );

int ldap_unbind_s( LDAP *ld );
```

Parameters are:

ld The session handle.

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

The ldap_unbind_ext(), ldap_unbind() and ldap_unbind_s() all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning. Note, however, that there is no server response to an LDAP unbind operation. All three of the unbind functions return LDAP_SUCCESS (or another LDAP result code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld.

The ldap_unbind() and ldap_unbind_s() functions behave identically. The ldap_unbind_ext() function allows server and client controls to be included explicitly, but note that since there is no server response to an unbind request there is no way to receive a response to a server control sent with an unbind request.

11.6. 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,
        const char      *base,
        int             scope,
        const char      *filter,
        char            **attrs,
        int             attrsonly,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
```

```
        struct timeval  *timeout,
        int             sizelimit,
        int             *msgidp
);

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

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

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

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

Parameters are:

ld — The session handle.

base — The dn of the entry at which to start the search. If NULL, a zero length DN is sent to the server.

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 [13], representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)" which matches all entries is to be used. Note that if the caller of the API is using LDAPv2, only a subset of the filter functionality described in [13] can be successfully used.

attrs — A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string LDAP_NO_ATTRS ("1.1") MAY be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string LDAP_ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.

attrsonly — A boolean value that MUST be zero if both attribute types and values are to be returned, and non-zero if only types are wanted.

timeout — For the ldap_search_st() function, this specifies the local search timeout value (if it is NULL, the timeout is infinite). If a zero timeout (where tv_sec and tv_usec are both zero) is passed, API implementations SHOULD return LDAP_PARAM_ERROR.

For the ldap_search_ext() and ldap_search_ext_s() functions, the timeout parameter specifies both the local search timeout value and the operation time limit that is sent to the server within the search request. Passing a NULL value for timeout causes the default timeout stored in the LDAP session handle (set by using ldap_set_option() with the LDAP_OPT_TIMELIMIT parameter) to be sent to the server with the request but an infinite local search

timeout to be used. If a zero timeout (where tv_sec and tv_usec are both zero) is passed in, API implementations SHOULD return LDAP_PARAM_ERROR. If a zero value for tv_sec is used but tv_usec is non-zero, an operation time limit of 1 SHOULD be passed to the LDAP server as the operation time limit. For other values of tv_sec, the tv_sec value itself SHOULD be passed to the LDAP server.

sizelimit For the ldap_search_ext() and ldap_search_ext_s() calls, this is a limit on the number of entries to return from the search. A value of LDAP_NO_LIMIT (0) means no limit. A value of LDAP_DEFAULT_SIZELIMIT (-1) means use the default timeout from the LDAP session handle (which is set by calling ldap_set_option() with the LDAP_OPT_SIZELIMIT parameter).

res For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call. If an API error occurs or no results are returned, *res is set to NULL.

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_search_ext() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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

```
LDAP_OPT_SIZELIMIT
LDAP_OPT_TIMELIMIT
LDAP_OPT_DEREF
```

These options are fully described in the earlier section "LDAP Session Handle Options."

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 result 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 result 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., can 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. The local search timeout is used to limit the amount of time the API implementation will wait for a search to complete. After the local search timeout expires, the API implementation will send an abandon operation to abort the search operation.

11.7. 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=*)" or NULL. attrs contains the list of attributes to return.

11.8. 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=*)" or NULL. attrs contains the list of attributes to return for each child entry.

11.9. 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,
        const char              *dn,
        const char              *attr,
        const struct berval     *bvalue,
        LDAPControl             **serverctrls,
        LDAPControl             **clientctrls,
        int                     *msgidp
);

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

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

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

Parameters are:

ld The session handle.

dn The name of the entry to compare against. If NULL, a zero length DN is sent to the server.

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, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_compare_ext() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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 result 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: one of the constants LDAP_COMPARE_TRUE or LDAP_COMPARE_FALSE if the operation was successful, or another LDAP result 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.

11.10. Modifying an entry

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

```
typedef union mod_vals_u {
        char            **modv_strvals;
        struct berval   **modv_bvals;
} mod_vals_u_t;

typedef struct ldapmod {
        int             mod_op;
        char            *mod_type;
        mod_vals_u_t    mod_vals;
} LDAPMod;
#define mod_values      mod_vals.modv_strvals
#define mod_bvalues     mod_vals.modv_bvals

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

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

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

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

Parameters are:

ld The session handle.

dn The name of the entry to modify. If NULL, a zero length DN is sent to the server.

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

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_modify_ext() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

The fields in the LDAPMod structure have the following meanings:

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

mod_type The type of the attribute to modify.

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

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

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

For LDAP_MOD_REPLACE modifications, the attribute 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 result 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 result 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.

11.11. Modifying the Name of an Entry

In LDAPv2, the ldap_modrdn(), ldap_modrdn_s(), ldap_modrdn2(), and ldap_modrdn2_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(), ldap_modrdn_s(), ldap_modrdn2(), and ldap_modrdn2_s() routines is deprecated.

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

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

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

```
        int ldap_modrdn(
                LDAP            *ld,
                const char      *dn,
                const char      *newrdn
        );
        int ldap_modrdn_s(
                LDAP            *ld,
                const char      *dn,
                const char      *newrdn
        );
        int ldap_modrdn2(
                LDAP            *ld,
                const char      *dn,
                const char      *newrdn,
                int             deleteoldrdn
        );
        int ldap_modrdn2_s(
                LDAP            *ld,
                const char      *dn,
                const char      *newrdn,
                int             deleteoldrdn
        );
```

Parameters are:

ld The session handle.

dn The name of the entry whose DN is to be changed. If NULL, a zero length DN is sent to the server.

newrdn The new RDN to give the entry.

newparent The new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is changed. The root DN

SHOULD be specified by passing a zero length string, "". The newparent parameter SHOULD always be NULL when using version 2 of the LDAP protocol; otherwise the server's behavior is undefined.

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

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_rename() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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 result 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 result 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.

11.12. 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,
        const char      *dn,
        LDAPMod         **attrs,
```

```
                LDAPControl     **serverctrls,
                LDAPControl     **clientctrls,
                int             *msgidp
        );

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

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

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

Parameters are:

ld — The session handle.

dn — The name of the entry to add. If NULL, a zero length DN is sent to the server.

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

serverctrls — List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls — List of client controls, or NULL if no client controls are to be used.

msgidp — This result parameter will be set to the message id of the request if the ldap_add_ext() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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 result 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 result 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.

11.13. 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,
        const char      *dn,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        int             *msgidp
);

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

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

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

Parameters are:

ld The session handle.

dn The name of the entry to delete. If NULL, a zero length DN is sent to the server.

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_delete_ext() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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 result 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 result 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.

11.14. 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,
        const char              *requestoid,
        const struct berval     *requestdata,
        LDAPControl             **serverctrls,
        LDAPControl             **clientctrls,
        int                     *msgidp
);

int ldap_extended_operation_s(
        LDAP                    *ld,
        const char              *requestoid,
        const struct berval     *requestdata,
        LDAPControl             **serverctrls,
        LDAPControl             **clientctrls,
        char                    **retoidp,
        struct berval           **retdatap
);
```

Parameters are:

ld The session handle.

requestoid The dotted-OID text string naming the request.

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

serverctrls List of LDAP server controls, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are

to be used.

msgidp This result parameter will be set to the message id of the request if the ldap_extended_operation() call succeeds. The value is undefined if a value other than LDAP_SUCCESS is returned.

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 an API error occurs or no OID is returned by the server, *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 an API error occurs or no data is returned by the server, *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 result 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 result 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.

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

12. 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, or NULL if no server controls are to be used.

clientctrls List of client controls, or NULL if no client controls are to be used.

ldap_abandon_ext() abandons the operation with message id msgid and returns the constant LDAP_SUCCESS if the abandon was successful or another LDAP result 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 does not accept client or server controls and it returns zero if the abandon was successful, -1 otherwise.

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

13. 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() can actually return a list or "chain" of result messages. The ldap_result() function only returns messages for a single request, so for all LDAP operations other than search only one result message is expected; that is, the only time the "result chain" can contain more than one message is if results from a search operation are returned. Once a chain of messages has been returned to the caller, it is no longer tied in any caller-visible way to the LDAP request that produced it. However, it MAY be tied to the session handle. Therefore, a chain of messages returned by calling ldap_result() or by calling a synchronous search routine will never be affected by subsequent LDAP API calls

except for ldap_msgfree() (which is used to dispose of a chain of messages) and the unbind calls (which dispose of a session handle): ldap_unbind(), ldap_unbind_s(), or ldap_unbind_ext(), or functions defined by extensions of this API.

ldap_msgfree() frees the result messages (possibly an entire chain of messages) obtained from a previous call to ldap_result() or from a call to a synchronous search routine.

ldap_msgtype() returns the type of an LDAP message.

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

```
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, the constant LDAP_RES_UNSOLICITED (0) if an unsolicited result is desired, or 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 messages retrieved so far are to be returned in the result chain.

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

specifies a polling behavior.

res For ldap_result(), a result parameter that will contain the result(s) of the operation. If an API error occurs or no results are returned, *res is set to NULL. For ldap_msgfree(), the result chain to be freed, obtained from a previous call to ldap_result(), ldap_search_s(), or ldap_search_st(). If res is NULL, nothing is done and ldap_msgfree() returns zero.

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 each message in the result chain pointed to by res and returns the type of the last message in the chain. If res is NULL, nothing is done and the value zero is returned.

ldap_msgtype() returns the type of the LDAP message it is passed as a parameter. The type 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, or -1 on error.

14. Handling Errors and Parsing Results

The following calls are used to extract information from results and handle errors returned by other LDAP API routines. Note that ldap_parse_sasl_bind_result() and ldap_parse_extended_result() must typically be used in addition to ldap_parse_result() to retrieve all the result information from SASL Bind and Extended Operations respectively.

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

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

int ldap_parse_extended_result(
        LDAP            *ld,
        LDAPMessage     *res,
        char            **retoidp,
        struct berval   **retdatap,
        int             freeit
);

#define LDAP_NOTICE_OF_DISCONNECTION    "1.3.6.1.4.1.1466.20036"

char *ldap_err2string( int err );
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

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

void ldap_perror( LDAP *ld, const 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 resultCode field from the LDAPMessage message. This is the indication from the server of the outcome of the operation. NULL SHOULD be passed to ignore this field.

matcheddnp If the server returned a matchedDN string to indicate how much of a name passed in a request was recognized, this result parameter will be filled in with that matchedDN string. Otherwise, this field will be set to NULL. NULL SHOULD be passed to ignore this field. The matched DN string SHOULD be freed by calling ldap_memfree() which is described later in this document. Note that the server may return a zero length matchedDN (in which case *matchednp is set to an allocated copy of "") which is different than not returning a value at all (in which case *matcheddnp is set to NULL).

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

referralsp This result parameter will be filled in with the contents of the referrals field from the LDAPMessage message, indicating zero or more alternate LDAP servers where the request is to be retried. The referrals array SHOULD be freed by calling ldap_value_free() which is described later in this document. NULL SHOULD be passed to ignore this field. If no referrals were returned, *referralsp is set to NULL.

serverctrlsp This result parameter will be filled in with an allocated array of controls copied out of the LDAPMessage message. If serverctrlsp is NULL, no controls are returned. The control array SHOULD be freed by calling ldap_controls_free() which was described earlier. If no controls were returned, *serverctrlsp is set to NULL.

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. If freeit is non-zero, the entire chain of messages represented by res is disposed of.

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 ber_bvfree(). NULL SHOULD be passed to ignore this field.

retoidp 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 SHOULD be passed to ignore this field. If no OID was returned, *retoidp is set to NULL. The LDAP_NOTICE_OF_DISCONNECTION macro is defined as a convenience for clients that wish to check an OID to see if it matches the one used for the unsolicited Notice of Disconnection (defined in RFC 2251[2] section 4.4.1).

retdatap For extended results, this result parameter 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 ber_bvfree(). NULL SHOULD be passed to ignore this field. If no data is returned, *retdatap is set to NULL.

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

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

The ldap_parse_result(), ldap_parse_sasl_bind_result(), and ldap_parse_extended_result() functions all skip over messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE when looking for a result message to parse. They return the constant LDAP_SUCCESS if the result was successfully parsed and another LDAP API result code if not. If a value other than LDAP_SUCCESS is returned, the values of all the result parameters are undefined. Note that the LDAP result code that indicates the outcome of the operation performed by the server is placed in the errcodep ldap_parse_result() parameter. If a chain of messages that contains more than one result message is passed to these routines they always operate on the first result in the chain.

ldap_err2string() is used to convert a numeric LDAP result code, as returned by ldap_parse_result(), ldap_parse_sasl_bind_result(), ldap_parse_extended_result() or one of the synchronous API operation calls, into an informative zero-terminated character string message describing the error. It returns a pointer to static data and it MUST NOT return NULL; the value returned is always a valid null-terminated "C" string.

15. 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 can 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.

If successful, ldap_count_messages() returns the number of messages contained in a chain of results; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_messages() call can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

16. 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 MAY 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.

16.1. Stepping Through a List of Entries or References

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

```
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().

ref The reference returned by a previous call to ldap_first_reference() or ldap_next_reference().

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

ldap_count_entries() returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid,

-1 is returned. The ldap_count_entries() call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

ldap_count_references() returns the number of references contained in a chain of search results; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_references() call can also be used to count the number of references that remain in a chain.

16.2. 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(). The BerElement type itself is an opaque structure that is described in more detail later in this document in the section "Encoded ASN.1 Value Manipulation".

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

is NULL, the ldap_memfree() call does nothing.

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 MAY be passed in subsequent calls to ldap_next_attribute() to step through the entry's attributes. After a set of calls to ldap_first_attribute() and ldap_next_attribute(), if ptr is non-NULL, it SHOULD be freed by calling ber_free(ptr, 0). Note that it is very important to pass the second parameter as 0 (zero) in this call, since the buffer associated with the BerElement does not point to separately allocated memory.

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

16.3. 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,
        const char      *attr
);

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

int ldap_count_values( char **vals );

int ldap_count_values_len( struct berval **vals );

void ldap_value_free( char **vals );
```

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

Parameters 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.

ldap_get_values() and ldap_get_values_len() return NULL if no values are found for attr or if an error occurs.

ldap_count_values() and ldap_count_values_len() return -1 if an error occurs such as the vals parameter being invalid.

If a NULL vals parameter is passed to ldap_value_free() or ldap_value_free_len(), nothing is done.

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

16.4. 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( const char *dn, int notypes );

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

char *ldap_dn2ufn( const 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(). If NULL, a zero length DN is used.

rdn The rdn to explode, such as returned in the components of the array returned by ldap_explode_dn(). If NULL, a zero length DN is used.

notypes A boolean parameter, if non-zero indicating that the dn or rdn components are to 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 newly allocated 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 [5]. The root DN is returned as a zero length string ("").

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

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

ldap_dn2ufn() converts the DN into the user friendly format described in [14]. The UFN returned is newly allocated space that SHOULD be freed by a call to ldap_memfree() when no longer in use.

16.5. Retrieving controls from an entry

ldap_get_entry_controls() is used to extract LDAP controls from an entry.

```
int ldap_get_entry_controls(
        LDAP            *ld,
        LDAPMessage     *entry,
        LDAPControl     ***serverctrlsp
);
```

Parameters are:

ld The session handle.

entry The entry to extract controls from, as returned by ldap_first_entry() or ldap_next_entry().

serverctrlsp This result parameter will be filled in with an allocated array of controls copied out of entry. The control array SHOULD be freed by calling ldap_controls_free(). If serverctrlsp is NULL, no controls are returned. If no controls were returned, *serverctrlsp is set to NULL.

ldap_get_entry_controls() returns an LDAP result code that indicates whether the reference could be successfully parsed (LDAP_SUCCESS if all goes well). If ldap_get_entry_controls() returns a value other than LDAP_SUCCESS, the value of the serverctrlsp output parameter is undefined.

16.6. Parsing References

ldap_parse_reference() is used to extract referrals and controls from a SearchResultReference message.

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

Parameters are:

ld The session handle.

ref The reference to parse, as returned by ldap_result(), ldap_first_reference(), or ldap_next_reference().

referralsp This result parameter will be filled in with an allocated array of character strings. The elements of the array are the referrals (typically LDAP URLs) contained in ref. The array SHOULD be freed when no longer in used by calling ldap_value_free(). If referralsp is NULL, the referral URLs are not returned. If no referrals were returned, *referralsp is set to NULL.

serverctrlsp This result parameter will be filled in with an allocated array of controls copied out of ref. The control array SHOULD be freed by calling ldap_controls_free(). If serverctrlsp is NULL, no controls are returned. If no controls were returned, *serverctrlsp is set to NULL.

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

ldap_parse_reference() returns an LDAP result code that indicates whether the reference could be successfully parsed (LDAP_SUCCESS if all goes well). If a value other than LDAP_SUCCESS is returned, the value of the referralsp and serverctrlsp result parameters are undefined.

17. 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.

Note that the functions defined in this section all provide a method for determining success or failure but generally do not provide access to specific error codes. Therefore, applications that require precise error information when encoding or decoding ASN.1 values SHOULD NOT use these functions.

17.1. BER Data Structures and Types

The following additional integral types are defined for use in manipulation of BER encoded ASN.1 values:

```
       typedef <impl_tag_t> ber_tag_t;     /* for BER tags */

       typedef <impl_int_t> ber_int_t;     /* for BER ints, enums, and Booleans
*/

       typedef <impl_unit_t> ber_uint_t;   /* unsigned equivalent of ber_uint_t
*/

       typedef <impl_slen_t> ber_slen_t;   /* signed equivalent of ber_len_t */
```

Note that the actual definition for these four integral types is implementation specific; that is, `<impl_tag_t>', `<impl_int_t>', `<impl_uint_t>', and `<impl_slen_t>' MUST each be replaced with an appropriate implementation-specific type.

The `ber_tag_t' type is an unsigned integral data type that is large enough to hold the largest BER tag supported by the API implementation. The width (number of significant bits) of `ber_tag_t' MUST be at least 32, greater than or equal to that of `unsigned int' (so that integer promotions won't promote it to `int'), and no wider than that of `unsigned long'.

The `ber_int_t' and `ber_uint_t' types are the signed and unsigned variants of an integral type that is large enough to hold integers for purposes of BER encoding and decoding. The width of `ber_int_t' MUST be at least 32 and no larger than that of `long'.

The `ber_slen_t' type is the signed variant of the `ber_len_t' integral type, i.e. if `ber_len_t' is unsigned long, then `ber_slen_t' is signed long. The `<impl_slen_t>' in the `ber_len_t' typedef MUST be replaced with an appropriate type. Note that `ber_slen_t' is not used directly in the C LDAP API but is provided for the convenience of application developers and for use by extensions to the API.

```
           typedef struct berval {
                   ber_len_t       bv_len;
                   char            *bv_val;
           } BerValue;
```

As defined earlier in the section "Common Data Structures", a berval structure contains an arbitrary sequence of bytes and an indication of its length. The bv_len element is an unsigned integer. The bv_val is not necessarily zero-terminated. Applications MAY allocate their own berval structures.

As defined earlier in the section "Common Data Structures", the BerElement structure is an opaque structure:

```
           typedef struct berelement BerElement;
```

It 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.

17.2. Memory Disposal and Utility Functions

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

ber_bvfree() frees a berval structure returned from this API. Both the bv->bv_val string and the berval structure itself are freed. If bv is NULL, this call does nothing.

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

ber_bvecfree() frees an array of berval structures returned from this API. Each of the berval structures in the array are freed using ber_bvfree(), then the array itself is freed. If bv is NULL, this call does nothing.

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

ber_bvdup() returns a copy of a berval structure. The bv_val field in the returned berval structure points to a different area of memory than the bv_val field in the bv argument. 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 SHOULD be freed by the caller. The second argument fbuf SHOULD always be set to 1 to ensure that the internal buffer used by the BER functions is freed as well as the BerElement container itself. If ber is NULL, this call does nothing.

17.3. 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 SHOULD 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 to be rearranged in tag and byte order or default values to be removed, so these functions are not sufficient for generating DER output as defined in X.509 and X.680. If the caller takes responsibility for ordering values of sets correctly and removing default values, DER output as defined in X.509 and X.680 can be produced.

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, const 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 and a non-negative number if successful. 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 a ber_tag_t specifying the tag to override the next element to be written to the ber. This works across calls. The integer tag value SHOULD contain the tag class, constructed bit, and tag value. For example, a tag of "[3]" for a constructed type is 0xA3U. All implementations MUST support tags that fit in a single octet (i.e., where the tag value is less than 32) and they MAY support larger tags.

'b' Boolean. The next argument is an ber_int_t, 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 0x01U is used for the element.

'e' Enumerated. The next argument is a ber_int_t, containing the enumerated value in the host's byte order. An enumerated element is output. If this format character is not preceded by the 't' format modifier, the tag 0x0AU is used for the element.

'i' Integer. The next argument is a ber_int_t, 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 0x02U is used for the element.

'B' Bitstring. The next two arguments are a char * pointer to the start of the bitstring, followed by a ber_len_t 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 0x03U is used for the element.

'X' Reserved and not to be used. In older revisions of this specification,

'n' Null. No argument is needed. An ASN.1 NULL element is output. If this format character is not preceded by the 't' format modifier, the tag 0x05U is used for the element.

'o' Octet string. The next two arguments are a char *, followed by a ber_len_t with the length of the string. The string MAY contain null bytes and are do not have to be zero-terminated. An octet string element is output, in primitive form. If this format character is not preceded by the 't' format modifier, the tag 0x04U is used for the element.

's' Octet string. The next argument is a char * pointing to a zero-terminated string. An octet string element in primitive form is output, which does not include the trailing '\0' (null) byte. If this format character is not preceded by the 't' format modifier, the tag 0x04U is used for the element.

'v' Several octet strings. The next argument is a char **, an array of char * pointers to zero-terminated strings. The last element in the array MUST be a NULL pointer. The octet strings do not include the trailing '\0' (null) byte. Note that a construct like '{v}' is used to get an actual SEQUENCE OF octet strings. The 't' format modifier cannot be used with this format character.

'V' Several octet strings. A NULL-terminated array of struct berval *'s is supplied. Note that a construct like '{V}' is used to get an actual SEQUENCE OF octet strings. The 't' format modifier cannot be used with this format character.

'{' Begin sequence. No argument is needed. If this format character is not preceded by the 't' format modifier, the tag 0x30U is used.

'}' End sequence. No argument is needed. The 't' format modifier

cannot be used with this format character.

'[' Begin set. No argument is needed. If this format character is not preceded by the 't' format modifier, the tag 0x31U is used.

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

Each use of a '{' format character SHOULD 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 ']' format characters.

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 structure, which SHOULD 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 is an error (i.e., -1 will be returned by ber_flatten() if this situation is exists).

17.4. 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(const char *s, ber_int_t val1, ber_int_t val2,
         struct berval **bvPtr)
{
     BerElement *ber;
     int rc = -1;

     *bvPtr = NULL;  /* in case of error */
```

```
            ber = ber_alloc_t(LBER_USE_DER);

            if (ber == NULL) return -1;

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

            if (val2 != 0) {
                    if (ber_printf(ber,"ti",(ber_tag_t)0x80,val2) == -1) {
                            goto done;
                    }
            }

            if (ber_printf(ber,"}") == -1) {
                    goto done;
            }

            rc = ber_flatten(ber,bvPtr);

   done:
            ber_free(ber,1);
            return rc;
      }
```

17.5. Decoding

The following two macros are available to applications: LBER_ERROR and LBER_DEFAULT. Both of these macros MUST be #define'd as ber_tag_t integral values that are treated as invalid tags by the API implementation. It is RECOMMENDED that the values of LBER_ERROR and LBER_DEFAULT be the same and that they be defined as values where all octets have the value 0xFF. ISO C guarantees that these definitions will work:

```
            #define LBER_ERROR   ((ber_tag_t)-1)
            #define LBER_DEFAULT ((ber_tag_t)-1)
```

The intent is that LBER_ERROR and LBER_DEFAULT are both defined as the integer value that has all octets set to 0xFF, as such a value is not a valid BER tag.

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

The ber_init function constructs a BerElement and returns a new BerElement containing a copy of the data in the bv argument. ber_init returns the NULL pointer on error.

```
ber_tag_t ber_scanf( BerElement *ber, const 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 SHOULD 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 different value on success. If an error occurred, the values of all the result parameters are undefined.

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 MUST be supplied. Memory is allocated, filled with the contents of the octet string, zero-terminated, and the pointer to the string is stored in the argument. The returned value SHOULD 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 MUST be supplied, which upon return points to an allocated struct berval containing the octet string and its length. ber_bvfree() SHOULD 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 a ber_int_t MUST be supplied. The ber_int_t 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.

'e' Enumerated. A pointer to a ber_int_t MUST be supplied. The enumerated 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 value of the enumerated value cannot be stored in a ber_int_t.

'i' Integer. A pointer to a ber_int_t MUST be supplied. The ber_int_t 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 a ber_int_t.

'B' Bitstring. A char ** argument MUST be supplied which will point to the allocated bits, followed by a ber_len_t * argument, which will point to the length (in bits) of the bitstring returned. ldap_memfree SHOULD be called to free the bitstring. The tag of the element MUST indicate the primitive form (constructed bit-strings are not supported) but is otherwise ignored during the decoding.

'n' Null. No argument is needed. The element is verified to have a zero-length value and is skipped. The tag is ignored.

't' Tag. A pointer to a ber_tag_t MUST be supplied. The ber_tag_t value stored will be the tag of the next element in the BerElement ber, represented so it can be written using the 't' format of ber_printf(). The decoding position within the ber argument is unchanged by this; that is, the fact that the tag has been retrieved does not affect future use of ber.

'v' Several octet strings. A char *** argument MUST be supplied, which upon return points to an allocated NULL-terminated array of char *'s containing the octet strings. NULL is stored if the sequence is empty. ldap_memfree SHOULD 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 *** MUST 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 needed.

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

'}' End sequence. No argument is needed.

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

']' End set. No argument is needed.

```
ber_tag_t ber_peek_tag( BerElement *ber,
        ber_len_t *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 decoding position within the ber argument is unchanged by this call; that is, the fact that ber_peek_tag() has been called does not affect future use of ber.

```
ber_tag_t ber_skip_tag( BerElement *ber, ber_len_t *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.

```
ber_tag_t ber_first_element( BerElement *ber,
        ber_len_t *lenPtr, char **opaquePtr );

ber_tag_t ber_next_element( BerElement *ber,
        ber_len_t *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.

17.6. 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 TAG_CONTROL_LIST 0xA0U /* context specific cons 0 */

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

        ber = ber_init(bv);
        if (ber == NULL) {
                fputs("ERROR ber_init failed\n", stderr);
                return -1;
        }

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

        if (res == LBER_ERROR) {
                fputs("ERROR ber_scanf failed\n", stderr);
                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((char *)attrs);

        if (ber_peek_tag(ber,&len) == TAG_CONTROL_LIST) {
                char *opaque;
                ber_tag_t tag;

                for (tag = ber_first_element(ber,&len,&opaque);
```

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

                           ber_len_t tlen;
                           ber_tag_t ttag;
                           char *type;
                           ber_int_t crit;
                           struct berval *value;

                           if (ber_scanf(ber,"{a",&type) == LBER_ERROR) {
                                   fputs("ERROR cannot parse type\n", stderr);
                                   break;
                           }
                           /* *** use type */
                           ldap_memfree(type);

                           ttag = ber_peek_tag(ber,&tlen);
                           if (ttag == 0x01U) {  /* boolean */
                                   if (ber_scanf(ber,"b",
                                                 &crit) == LBER_ERROR) {
                                           fputs("ERROR cannot parse crit\n",
                                               stderr);
                                           rc = -1;
                                           break;
                                   }
                           } else if (ttag == 0x04U) { /* octet string */
                                   crit = 0;
                           } else {
                                   fputs("ERROR extra field in controls\n",
                                       stderr );
                                   break;
                           }

                           if (ber_scanf(ber,"O}",&value) == LBER_ERROR) {
                                   fputs("ERROR cannot parse value\n", stderr);
                                   rc = -1;
                                   break;
                           }
                           /* *** use value */
                           ber_bvfree(value);
                   }
           }

           if ( rc == 0 ) {        /* no errors so far */
                   if (ber_scanf(ber,"}") == LBER_ERROR) {
                           rc = -1;
                   }
           }
```

```
        ber_free(ber,1);

        return rc;
    }
```

18. Security Considerations

LDAPv2 supports security through protocol-level authentication using clear-text passwords. LDAPv3 adds support for SASL [12] (Simple Authentication Security Layer) methods. LDAPv3 also supports operation over a secure transport layer using Transport Layer Security TLS [9]. 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.

19. Acknowledgements

Many members of the IETF ASID and LDAPEXT working groups as well as members of the Internet at large have provided useful comments and suggestions that have been incorporated into this document. Chris Weider deserves special mention for his contributions as co-author of earlier revisions of this document.

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

20. Copyright

languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

21. Bibliography

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[2] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.

[3] M. Wahl, A. Coulbeck, T. Howes, S. Kille, W. Yeong, C. Robbins, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

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

[5] M. Wahl, S. Kille, T. Howes, "Lightweight Directory Access Protocol (v3): A UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

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

[7] K. Simonsen, "Character Mnemonics and Character Sets," RFC 1345, June 1992.

[8] "Programming Languages - C", ANSI/ISO Standard 9899, revised 1997.

[9] J. Hodges, R. Morgan, M. Wahl, "Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security", INTERNET-DRAFT (work in progress) <draft-ietf-ldapext-ldapv3-tls-05.txt>, June 1999.

[10] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture," RFC 1884, December 1995.

[11] A. Herron, T. Howes, M. Wahl, A. Anantha, "LDAP Control Extension for Server Side Sorting of Search Results", INTERNET-DRAFT (work in progress) <draft-ietf-ldapext-sorting-02.txt>, 5 April 1999.

[12] J. Meyers, "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[13] T. Howes, "The String Representation of LDAP Search Filters," RFC 2254, December 1997.

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

22. Authors' Addresses

Mark Smith (document editor)
Netscape Communications Corp.
901 San Antonio Rd.
Palo Alto, CA 94303-4900
Mail Stop SCA17 - 201
USA
+1 650 937-3477
mcs@netscape.com

Tim Howes
Loudcloud, Inc.
599 N. Mathilda Avenue
Sunnyvale, CA 94085
USA
+1 408 744-7300
howes@loudcloud.com

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

Mark Wahl
Sun Microsystems, Inc.
8911 Capital of Texas Hwy, Suite 4140
Austin, TX 78759
USA
+1 626 919 3600
Mark.Wahl@sun.com

Anoop Anantha
Microsoft Corp.
1 Microsoft Way
Redmond, WA 98052
USA
+1 425 882-8080
anoopa@microsoft.com

23. Appendix A - Sample C LDAP API Code

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

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

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

        /* authenticate as nobody */
        if (( rc = ldap_simple_bind_s( ld, NULL, NULL )) != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_simple_bind_s: %s\n",
                    ldap_err2string( rc ));
                ldap_unbind( ld );
                return 1;
        }

        /* search for entries with cn of "Babs Jensen", return all attrs  */
        if (( rc = ldap_search_s( ld, "o=University of Michigan, c=US",
            LDAP_SCOPE_SUBTREE, "(cn=Babs Jensen)", NULL, 0, &res ))
            != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_search_s: %s\n",
                    ldap_err2string( rc ));
                if ( res == NULL ) {
                        ldap_unbind( ld );
                        return 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( "\tattribute: %s\n", a );

                            /* print each value */
                            vals = ldap_get_values( ld, e, a );
                            for ( i = 0; vals[i] != NULL; i++ ) {
                                    printf( "\t\tvalue: %s\n", vals[i] );
                            }
                            ldap_value_free( vals );
                            ldap_memfree( a );
                    }
                    if ( ptr != NULL ) {
                            ber_free( ptr, 0 );
                    }
            }
            /* free the search results */
            ldap_msgfree( res );

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

            return 0;
    }
```

24. Appendix B - Namespace Consumed By This Specification

The following 2 prefixes are used in this specification to name functions:

- ldap_
- ber_

The following 6 prefixes are used in this specification to name structures, unions, and typedefs:

- ldap
- LDAP
- mod_vals_u
- ber
- Ber

timeval

The following 3 prefixes are used in this specification to name #defined macros:

LDAP
LBER_
mod_

25. Appendix C - Summary of Requirements for API Extensions

As the LDAP protocol is extended, this C LDAP API will need to be extended as well. For example, an LDAPv3 control extension has already been defined for server-side sorting of search results [7]. This appendix summarizes the requirements for extending this API.

25.1. Compatibility

Extensions to this document SHOULD NOT, by default, alter the behavior of any of the APIs specified in this document. If an extension optionally changes the behavior of any existing C LDAP API function calls, the behavior change MUST be well documented. If an extension that operates on an LDAP session affects a chain of messages that was previously obtained by a call to ldap_result() or by calling a synchronous search routine, this MUST be well documented.

25.2. Style

Extensions to this API SHOULD follow the general style and naming conventions used in this document. For example, function names SHOULD start with "ldap_" or "ber_" and consist entirely of lowercase letters, digits, and underscore ('_') characters. It is RECOMMENDED that private and experimental extensions use only the following prefixes for macros, types, and function names:

LDAP_X_
LBER_X_
ldap_x_
ber_x_

and that these prefixes not be used by standard extensions.

25.3. Dependence on Externally Defined Types

Extensions to this API SHOULD minimize dependencies on types and macros that are defined in system headers and generally use only intrinsic types that are part of the C language, types defined in this specification, or types defined in the extension document itself.

25.4. Compile Time Information

Extensions to this API SHOULD conform to the requirements contained in the "Retrieving Information at Compile Time" section of this document. That is, extensions SHOULD define a macro of the form:

```
#define LDAP_API_FEATURE_x level
```

so that applications can detect the presence or absence of the extension at compile time and also test the version or level of the extension provided by an API implementation.

25.5. Runtime Information

Extensions to this API SHOULD conform to the requirements contained in the "Retrieving Information During Execution" section of this document. That is, each extension SHOULD be given a character string name and that name SHOULD appear in the ldapai_extensions array field of the LDAPAPI-Info structure following a successful call to ldap_get_option() with an option parameter value of LDAP_OPT_API_INFO. In addition, information about the extension SHOULD be available via a call to ldap_get_option() with an option parameter value of LDAP_OPT_API_FEATURE_INFO.

25.6. Values Used for Session Handle Options

Extensions to this API that add new session options (for use with the ldap_get_option() and ldap_set_option() functions) SHOULD meet the requirements contained in the last paragraph of the "LDAP Session Handle Options" section of this document. Specifically, standards track documents MUST use values for option macros that are between 0x1000 and 0x3FFF inclusive and private and experimental extensions MUST use values for the option macros that are between 0x4000 and 0x7FFF inclusive.

26. Appendix D - Known Incompatibilities with RFC 1823

This appendix lists known incompatibilities between this API specification and the one contained in RFC 1823, beyond the additional API functions added in support of LDAPv3.

26.1. Opaque LDAP Structure

In RFC 1823, some fields in the LDAP structure were exposed to application programmers. To provide a cleaner interface and to make it easier for implementations to evolve over time without sacrificing binary compatibility with older applications, the LDAP structure is now entirely opaque. The new ldap_set_option() and ldap_get_option() calls can be

used to manipulate per-session and global options.

26.2. Additional Result Codes

The following new result code macros were introduced to support LDAPv3:

```
LDAP_REFERRAL
LDAP_ADMINLIMIT_EXCEEDED
LDAP_UNAVAILABLE_CRITICAL_EXTENSION
LDAP_CONFIDENTIALITY_REQUIRED
LDAP_SASL_BIND_IN_PROGRESS
LDAP_AFFECTS_MULTIPLE_DSAS
LDAP_CONNECT_ERROR
LDAP_NOT_SUPPORTED
LDAP_CONTROL_NOT_FOUND
LDAP_NO_RESULTS_RETURNED
LDAP_MORE_RESULTS_TO_RETURN
LDAP_CLIENT_LOOP
LDAP_REFERRAL_LIMIT_EXCEEDED
```

26.3. Freeing of String Data with ldap_memfree()

All strings received from the API (e.g., those returned by the ldap_get_dn() or ldap_dn2ufn() functions) SHOULD be freed by calling ldap_memfree() not free(). RFC 1823 did not define an ldap_memfree() function.

26.4. Changes to ldap_result()

The meaning of the all parameter to ldap_result has changed slightly. Nonzero values from RFC 1823 correspond to LDAP_MSG_ALL (0x01). There is also a new possible value, LDAP_MSG_RECEIVED (0x02).

The result type LDAP_RES_MODDN is now returned where RFC 1823 returned LDAP_RES_MODRDN. The actual value for these two macros is the same (0x6D).

26.5. Changes to ldap_first_attribute() and ldap_next_attribute

Each non-NULL return value SHOULD be freed by calling ldap_memfree() after use. In RFC 1823, these two functions returned a pointer to a per-session buffer, which was not very thread-friendly.

After the last call to ldap_first_attribute() or ldap_next_attribute(), the value set in the ptr parameter SHOULD be freed by calling ber_free(

ptr, 0). RFC 1823 did not mention that the ptr value SHOULD be freed.

The type of the ptr parameter was changed from void * to BerElement *.

26.6. Changes to ldap_modrdn() and ldap_modrdn_s() Functions

In RFC 1823, the ldap_modrdn() and ldap_modrdn_s() functions include a parameter called deleteoldrdn. This does not match the great majority of implementations, so in this specification the deleteoldrdn parameter was removed from ldap_modrdn() and ldap_modrdn_s(). Two additional functions that support deleteoldrdn and are widely implemented as well were added to this specification: ldap_modrdn2() and ldap_modrdn2_s().

26.7. Changes to the berval structure

In RFC 1823, the bv_len element of the berval structure was defined as an `unsigned long'. In this specification, the type is implementation-specific, although it MUST be an unsigned integral type that is at least **32 bits in size.** See the appendix "Data Types and Legacy Implementations" for additional considerations.

26.8. API Specification Clarified

RFC 1823 left many things unspecified, including behavior of various memory disposal functions when a NULL pointer is presented, requirements for headers, values of many macros, and so on. This specification is more complete and generally tighter than the one in RFC 1823.

26.9. Deprecated Functions

A number of functions that are in RFC 1823 are labeled as "deprecated" in this specification. In most cases, a replacement that provides equivalent functionality has been defined. The deprecated functions are:

ldap_bind()
: Use ldap_simple_bind() or ldap_sasl_bind() instead.

ldap_bind_s()
: Use ldap_simple_bind_s() or ldap_sasl_bind_s() instead.

ldap_kerberos_bind() and ldap_kerberos_bind_s()
: No equivalent functions are provided.

ldap_modrdn() and ldap_modrdn2()
Use ldap_rename() instead.

ldap_modrdn_s() and ldap_modrdn2_s()
Use ldap_rename_s() instead.

ldap_open()
Use ldap_init() instead.

ldap_perror()
Use ldap_get_option(ld, LDAP_OPT_RESULT_CODE, &rc) followed by fprintf(stderr, "%s: %s", msg, ldap_err2string(rc)) instead.

ldap_result2error()
Use ldap_parse_result() instead.

27. Appendix E - Data Types and Legacy Implementations

The data types associated with the length of a ber value (ber_len_t), and the tag (ber_tag_t) have been defined in this specification as unsigned integral types of implementation-specific size. The data type used for encoding and decoding ber integer, enumerated, and boolean values has been defined in this specification as a signed integral type of implementation-specific size. This was done so that source and binary compatibility of the C LDAP API can be maintained across ILP32 environments (where int, long, and pointers are all 32 bits in size) and LP64 environments (where ints remain 32 bits but longs and pointers grow to 64 bits).

In older implementations of the C LDAP API, such as those based on RFC 1823, implementors may have chosen to use an `unsigned long' for length and tag values. If a long data type was used for either of these items, a port of an application to a 64-bit operating system using the LP64 data model would find the size of the types used by the C LDAP API to increase. Also, if the legacy implementation had chosen to implement the tag and types as an unsigned int, adoption of a specification that mandated use of unsigned longs would cause a source incompatibility in an LP64 application. By using implementation-specific data types, the C LDAP API implementation is free to choose the correct data type and the ability to maintain source compatibility.

For example, suppose a legacy implementation chose to define the return value of ber_skip_tag() as an unsigned long but wishes to have the library return a 32-bit quantity in both ILP32 and LP64 data models. The following typedefs for ber_tag_t will provide a fixed sized data structure while preserving existing ILP32 source -- all without

```
generating compiler warnings:
           #include <limits.h>     /* provides UINT_MAX in ISO C */
           #if UINT_MAX >= 0xffffffffU
               typedef unsigned int ber_tag_t;
           #else
               typedef unsigned long ber_tag_t;
           #endif
```

Similar code can be used to define appropriate ber_len_t, ber_int_t, ber_slen_t and ber_uint_t types.

28. Appendix F - Changes Made Since Last Document Revision

The previous version of this document was draft-ietf-ldapext-ldap-c-api-04.txt, dated 8 October 1999. This appendix lists all of the changes made to that document to produce this one.

28.1. API Changes

"Header Requirements" section: added requirement that the simple program provided must execute as well as compile without errors.

"LDAP Session Handle Options" section: changed the name of the LDAP_OPT_ERROR_NUMBER option to LDAP_OPT_RESULT_CODE. Allow LDAP_OPT_ON to be defined as an implementation specific value (to avoid problems on architectures where the value ((void *)1) is not usable).

"Initializing an LDAP Session" section: allow use of the value zero for the `portno' parameter to mean "use port 389."

"Searching" section: added LDAP_DEFAULT_SIZELIMIT (-1) to allow application programmers to use the sizelimit from the LDAP session handle with ldap_search_ext() and ldap_search_ext_s().

"Modifying an entry" section: moved mod_vals union out of LDAPMod and added mod_vals_u_t typedef so users of the API can declare variables using the union type. "Handling Errors and Parsing Results" section: added text to require that ldap_err2string() MUST NOT return NULL.

"A Client Control That Governs Referral Processing" section: modified the text to specify that a ber_uint_t value should be used to hold the flags.

28.2. Editorial Changes and Clarifications

"Overview of LDAP API Use and General Requirements" section: added text to clarify our use of the term "asynchronous."

"Retrieving Information During Execution" section: added text describing the `ldapai_vendor_name' and `ldapai_vendor_version' fields (text was accidently deleted during a previous round of edits).

"LDAP Session Handle Options" section: improved the text that describes the LDAP_OPT_TIMELIMIT, LDAP_OPT_SIZELIMIT, and LDAP_OPT_RESULT_CODE options. Provided details and an example of the correct LDAP_OPT_HOST_NAME string to return when the `portno' passed to ldap_init() is not zero or 389.

"Result Codes" section: renamed section (was "LDAP Error Codes").

"Authenticating to the directory" section: clarified that the `dn', `cred', and `passwd' parameters can be NULL. Added text indicate that the `servercredp' is set to NULL if an API error occurs.

"Performing LDAP Operations" section: replaced "All functions take a session handle" with "Most functions...."

"Search" section: removed the detailed discussion of the session handle options (already covered in the "Retrieving Information During Execution" section). Also removed the word "global" when discussing the session default value for the `timeout' parameter. Also clarified that a NULL `base' parameter means use a zero-length string for the base DN.

"Comparing a Value Against an Entry" section: corrected the "successful" return codes for ldap_compare_ext_s() and ldap_compare_s() (was LDAP_SUCCESS; changed to LDAP_COMPARE_TRUE or LDAP_COMPARE_FALSE).

"Extended Operations" section: added text to indicate that the `retoidp' and `retdatap' result parameters are set to NULL if an API error occurs in ldap_extended_operation_s().

"Handling Errors and Parsing Results" section: added text to say that the `matcheddnp' result parameter will be set to NULL if the server does not return a matched DN string. Added text to indicate that serverctrlsp can be NULL. Added text to indicate that *retoidpp, *retdatap, *referralsp, and *serverctrlsp will be set to NULL if no items of that type are returned. Removed specific reference to LDAP_NO_SUCH_OBJECT result code when discussing the `matcheddnp' result parameter and added clarifying note about "" vs. NULL.

"Parsing References" section: added text to indicate that *referralsp, and *serverctrlsp will be set to NULL if no items of that type are returned.

"Obtaining Results and Peeking Inside LDAP Messages" section: added text to say that LDAPMessage chains MAY be tied to a session handle.

"BER Data Structures and Types" section: removed note about ber_uint_t not being used in this document (it is now). Changed text to simplify the description of ber_slen_t. Removed misleading sentence about the width of ber_uint_t.

"Encoded ASN.1 Value Manipulation / Encoding" section: added note that 'X' is reserved. Also fixed a few small bugs in the example code.

"Encoded ASN.1 Value Manipulation / Decoding" section: clarified the requirements for LBER_ERROR and LBER_DEFAULT (expressed using octets instead of bits). Also fixed a few small bugs in the example code.

Added the following text to all descriptions of the `serverctrls' and `clientctrls' parameters: ", or NULL if no <server/client> controls are to be used."

Added the following text to the description of all `dn' and `rdn' parameters: "If NULL, a zero length DN is sent to the server."

Replaced many occurrences of the phrase "error code" with "result code" throughout the document.

Added text to indicate that the value of the `msgidp' result parameter is undefined if an error occurs in the following functions: ldap_sasl_bind(), ldap_search_ext(), ldap_compare_ext(), ldap_modify_ext(), ldap_add_ext(), ldap_delete_ext(), ldap_extended_operation().

Added text to indicate that the `res' result parameter is set to NULL if an API error occurs in the following functions: ldap_result(), ldap_search_s(), ldap_search_st().

Added text to indicate that all result parameters have undefined values if an API error is returned by the following functions: ldap_parse_result(), ldap_parse_sasl_bind_result(), ldap_parse_extended_result(), ldap_parse_reference(), ber_scanf().

Added angle brackets around ficticious impl_XXX_t types to make it more obvious that these are not real "C" types, e.g., typedef <impl_len_t> ber_len_t'.

Appendix B: Added mod_vals_u and removed PLDAP from the struct, unions, and typedefs prefix list.

Appendix C: Added note in "Compatibility" section about extensions possible affecting chains of messages and the fact that that must be well documented. Appendix D: Improved text for ldap_perror() (what to use instead).

"Authors" section: updated contact information for Mark Smith, Tim Howes, and Mark Wahl.

Fixed a few obvious typos, improved indentation, added missing blank lines, and so on.