Network Working Group
Request for Comments: 2251
Category: Standards Track

M. Wahl
Critical Angle Inc.
T. Howes
Netscape Communications Corp.
S. Kille
Isode Limited

December 1997

Lightweight Directory Access Protocol (v3)

1. Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

IESG Note

This document describes a directory access protocol that provides both read and update access. Update access requires secure authentication, but this document does not mandate implementation of any satisfactory authentication mechanisms.

In accordance with RFC 2026, section 4.4.1, this specification is being approved by IESG as a Proposed Standard despite this limitation, for the following reasons:

a. to encourage implementation and interoperability testing of these protocols (with or without update access) before they are deployed, and

b. to encourage deployment and use of these protocols in read-only applications. (e.g. applications where LDAPv3 is used as a query language for directories which are updated by some secure mechanism other than LDAP), and

c. to avoid delaying the advancement and deployment of other Internet standards-track protocols which require the ability to query, but not update, LDAPv3 directory servers.

Readers are hereby warned that until mandatory authentication mechanisms are standardized, clients and servers written according to this specification which make use of update functionality are UNLIKELY TO INTEROPERATE, or MAY INTEROPERATE ONLY IF AUTHENTICATION IS REDUCED TO AN UNACCEPTABLY WEAK LEVEL.

Implementors are hereby discouraged from deploying LDAPv3 clients or servers which implement the update functionality, until a Proposed Standard for mandatory authentication in LDAPv3 has been approved and published as an RFC.

Table of Contents

2. Abstract

The protocol described in this document is designed to provide access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). This protocol is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. When used with a directory supporting the X.500 protocols, it is intended to be a complement to the X.500 DAP.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in RFC 2119 [10].

Key aspects of this version of LDAP are:

- All protocol elements of LDAPv2 (RFC 1777) are supported. The protocol is carried directly over TCP or other transport, bypassing much of the session/presentation overhead of X.500 DAP.

- Most protocol data elements can be encoded as ordinary strings (e.g., Distinguished Names).

- Referrals to other servers may be returned.

- SASL mechanisms may be used with LDAP to provide association security services.

- Attribute values and Distinguished Names have been internationalized through the use of the ISO 10646 character set.

- The protocol can be extended to support new operations, and controls may be used to extend existing operations.

- Schema is published in the directory for use by clients.

3. Models

Interest in X.500 [1] directory technologies in the Internet has led to efforts to reduce the high cost of entry associated with use of these technologies. This document continues the efforts to define directory protocol alternatives, updating the LDAP [2] protocol specification.

3.1. Protocol Model

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, a client transmits a protocol request describing the operation to be performed to a server. The server is then responsible for performing the necessary operation(s) in the directory. Upon completion of the operation(s), the server returns a response containing any results or errors to the requesting client.

In keeping with the goal of easing the costs associated with use of the directory, it is an objective of this protocol to minimize the complexity of clients so as to facilitate widespread deployment of applications capable of using the directory.

Note that although servers are required to return responses whenever such responses are defined in the protocol, there is no requirement for synchronous behavior on the part of either clients or servers. Requests and responses for multiple operations may be exchanged between a client and server in any order, provided the client eventually receives a response for every request that requires one.

In LDAP versions 1 and 2, no provision was made for protocol servers returning referrals to clients. However, for improved performance and distribution this version of the protocol permits servers to return to clients referrals to other servers. This allows servers to offload the work of contacting other servers to progress operations.

Note that the core protocol operations defined in this document can be mapped to a strict subset of the X.500(1997) directory abstract service, so it can be cleanly provided by the DAP. However there is not a one-to-one mapping between LDAP protocol operations and DAP operations: server implementations acting as a gateway to X.500 directories may need to make multiple DAP requests.

3.2. Data Model

This section provides a brief introduction to the X.500 data model, as used by LDAP.

The LDAP protocol assumes there are one or more servers which jointly provide access to a Directory Information Tree (DIT). The tree is made up of entries. Entries have names: one or more attribute values from the entry form its relative distinguished name (RDN), which MUST be unique among all its siblings. The concatenation of the relative distinguished names of the sequence of entries from a particular entry to an immediate subordinate of the root of the tree forms that entry's Distinguished Name (DN), which is unique in the tree. An example of a Distinguished Name is

CN=Steve Kille, O=Isode Limited, C=GB

Some servers may hold cache or shadow copies of entries, which can be used to answer search and comparison queries, but will return referrals or contact other servers if modification operations are requested.

Servers which perform caching or shadowing MUST ensure that they do not violate any access control constraints placed on the data by the originating server.

The largest collection of entries, starting at an entry that is mastered by a particular server, and including all its subordinates and their subordinates, down to the entries which are mastered by different servers, is termed a naming context. The root of the DIT is a DSA-specific Entry (DSE) and not part of any naming context: each server has different attribute values in the root DSE. (DSA is an X.500 term for the directory server).

3.2.1. Attributes of Entries

Entries consist of a set of attributes. An attribute is a type with one or more associated values. The attribute type is identified by a short descriptive name and an OID (object identifier). The attribute type governs whether there can be more than one value of an attribute of that type in an entry, the syntax to which the values must conform, the kinds of matching which can be performed on values of that attribute, and other functions.

An example of an attribute is "mail". There may be one or more values of this attribute, they must be IA5 (ASCII) strings, and they are case insensitive (e.g. "foo@bar.com" will match "FOO@BAR.COM").

Schema is the collection of attribute type definitions, object class definitions and other information which a server uses to determine how to match a filter or attribute value assertion (in a compare operation) against the attributes of an entry, and whether to permit add and modify operations. The definition of schema for use with LDAP is given in [5] and [6]. Additional schema elements may be defined in other documents.

Each entry MUST have an objectClass attribute. The objectClass attribute specifies the object classes of an entry, which along with the system and user schema determine the permitted attributes of an entry. Values of this attribute may be modified by clients, but the objectClass attribute cannot be removed. Servers may restrict the modifications of this attribute to prevent the basic structural class of the entry from being changed (e.g. one cannot change a person into a country). When creating an entry or adding an objectClass value to an entry, all superclasses of the named classes are implicitly added as well if not already present, and the client must supply values for any mandatory attributes of new superclasses.

Some attributes, termed operational attributes, are used by servers for administering the directory system itself. They are not returned in search results unless explicitly requested by name. Attributes which are not operational, such as "mail", will have their schema and syntax constraints enforced by servers, but servers will generally not make use of their values.

Servers MUST NOT permit clients to add attributes to an entry unless those attributes are permitted by the object class definitions, the schema controlling that entry (specified in the subschema - see below), or are operational attributes known to that server and used for administrative purposes. Note that there is a particular objectClass 'extensibleObject' defined in [5] which permits all user attributes to be present in an entry.

Entries MAY contain, among others, the following operational attributes, defined in [5]. These attributes are maintained automatically by the server and are not modifiable by clients:

- creatorsName: the Distinguished Name of the user who added this entry to the directory.

- createTimestamp: the time this entry was added to the directory.

- modifiersName: the Distinguished Name of the user who last modified this entry.

- modifyTimestamp: the time this entry was last modified.

- subschemaSubentry: the Distinguished Name of the subschema entry (or subentry) which controls the schema for this entry.

3.2.2. Subschema Entries and Subentries

Subschema entries are used for administering information about the directory schema, in particular the object classes and attribute types supported by directory servers. A single subschema entry contains all schema definitions used by entries in a particular part of the directory tree.

Servers which follow X.500(93) models SHOULD implement subschema using the X.500 subschema mechanisms, and so these subschemas are not ordinary entries. LDAP clients SHOULD NOT assume that servers implement any of the other aspects of X.500 subschema. A server which masters entries and permits clients to modify these entries MUST implement and provide access to these subschema entries, so that its clients may discover the attributes and object classes which are permitted to be present. It is strongly recommended that all other servers implement this as well.

The following four attributes MUST be present in all subschema entries:

- cn: this attribute MUST be used to form the RDN of the subschema entry.

- objectClass: the attribute MUST have at least the values "top" and "subschema".

- objectClasses: each value of this attribute specifies an object class known to the server.

- attributeTypes: each value of this attribute specifies an attribute type known to the server.

These are defined in [5]. Other attributes MAY be present in subschema entries, to reflect additional supported capabilities.

These include matchingRules, matchingRuleUse, dITStructureRules, dITContentRules, nameForms and ldapSyntaxes.

Servers SHOULD provide the attributes createTimestamp and modifyTimestamp in subschema entries, in order to allow clients to maintain their caches of schema information.

Clients MUST only retrieve attributes from a subschema entry by requesting a base object search of the entry, where the search filter is "(objectClass=subschema)". (This will allow LDAPv3 servers which gateway to X.500(93) to detect that subentry information is being requested.)

3.3. Relationship to X.500

This document defines LDAP in terms of X.500 as an X.500 access mechanism. An LDAP server MUST act in accordance with the X.500(1993) series of ITU recommendations when providing the service. However, it is not required that an LDAP server make use of any X.500 protocols in providing this service, e.g. LDAP can be mapped onto any other directory system so long as the X.500 data and service model as used in LDAP is not violated in the LDAP interface.

3.4. Server-specific Data Requirements

An LDAP server MUST provide information about itself and other information that is specific to each server. This is represented as a group of attributes located in the root DSE (DSA-Specific Entry), which is named with the zero-length LDAPDN. These attributes are retrievable if a client performs a base object search of the root with filter "(objectClass=*)", however they are subject to access control restrictions. The root DSE MUST NOT be included if the client performs a subtree search starting from the root.

Servers may allow clients to modify these attributes.

The following attributes of the root DSE are defined in section 5 of [5]. Additional attributes may be defined in other documents.

- namingContexts: naming contexts held in the server. Naming contexts are defined in section 17 of X.501 [6].

- subschemaSubentry: subschema entries (or subentries) known by this server.

- altServer: alternative servers in case this one is later unavailable.

- supportedExtension: list of supported extended operations.

- supportedControl: list of supported controls.

- supportedSASLMechanisms: list of supported SASL security features.

- supportedLDAPVersion: LDAP versions implemented by the server.

If the server does not master entries and does not know the locations of schema information, the subschemaSubentry attribute is not present in the root DSE. If the server masters directory entries under one or more schema rules, there may be any number of values of the subschemaSubentry attribute in the root DSE.

4. Elements of Protocol

The LDAP protocol is described using Abstract Syntax Notation 1 (ASN.1) [3], and is typically transferred using a subset of ASN.1 Basic Encoding Rules [11]. In order to support future extensions to this protocol, clients and servers MUST ignore elements of SEQUENCE encodings whose tags they do not recognize.

Note that unlike X.500, each change to the LDAP protocol other than through the extension mechanisms will have a different version number. A client will indicate the version it supports as part of the bind request, described in section 4.2. If a client has not sent a bind, the server MUST assume that version 3 is supported in the client (since version 2 required that the client bind first).

Clients may determine the protocol version a server supports by reading the supportedLDAPVersion attribute from the root DSE. Servers which implement version 3 or later versions MUST provide this attribute. Servers which only implement version 2 may not provide this attribute.

4.1. Common Elements

This section describes the LDAPMessage envelope PDU (Protocol Data Unit) format, as well as data type definitions which are used in the protocol operations.

4.1.1. Message Envelope

For the purposes of protocol exchanges, all protocol operations are encapsulated in a common envelope, the LDAPMessage, which is defined as follows:

```
LDAPMessage ::= SEQUENCE {
```

```
                messageID       MessageID,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindResponse    BindResponse,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
                        abandonRequest  AbandonRequest,
                        extendedReq     ExtendedRequest,
                        extendedResp    ExtendedResponse },
                 controls       [0] Controls OPTIONAL }

        MessageID ::= INTEGER (0 .. maxInt)

        maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

The function of the LDAPMessage is to provide an envelope containing common fields required in all protocol exchanges. At this time the only common fields are the message ID and the controls.

If the server receives a PDU from the client in which the LDAPMessage SEQUENCE tag cannot be recognized, the messageID cannot be parsed, the tag of the protocolOp is not recognized as a request, or the encoding structures or lengths of data fields are found to be incorrect, then the server MUST return the notice of disconnection described in section 4.4.1, with resultCode protocolError, and immediately close the connection. In other cases that the server cannot parse the request received by the client, the server MUST return an appropriate response to the request, with the resultCode set to protocolError.

If the client receives a PDU from the server which cannot be parsed, the client may discard the PDU, or may abruptly close the connection.

The ASN.1 type Controls is defined in section 4.1.12.

4.1.1.1. Message ID

All LDAPMessage envelopes encapsulating responses contain the messageID value of the corresponding request LDAPMessage.

The message ID of a request MUST have a value different from the values of any other requests outstanding in the LDAP session of which this message is a part.

A client MUST NOT send a second request with the same message ID as an earlier request on the same connection if the client has not received the final response from the earlier request. Otherwise the behavior is undefined. Typical clients increment a counter for each request.

A client MUST NOT reuse the message id of an abandonRequest or of the abandoned operation until it has received a response from the server for another request invoked subsequent to the abandonRequest, as the abandonRequest itself does not have a response.

4.1.2. String Types

The LDAPString is a notational convenience to indicate that, although strings of LDAPString type encode as OCTET STRING types, the ISO 10646 [13] character set (a superset of Unicode) is used, encoded following the UTF-8 algorithm [14]. Note that in the UTF-8 algorithm characters which are the same as ASCII (0x0000 through 0x007F) are represented as that same ASCII character in a single byte. The other byte values are used to form a variable-length encoding of an arbitrary character.

```
LDAPString ::= OCTET STRING
```

The LDAPOID is a notational convenience to indicate that the permitted value of this string is a (UTF-8 encoded) dotted-decimal representation of an OBJECT IDENTIFIER.

```
LDAPOID ::= OCTET STRING
```

For example,

```
1.3.6.1.4.1.1466.1.2.3
```

4.1.3. Distinguished Name and Relative Distinguished Name

An LDAPDN and a RelativeLDAPDN are respectively defined to be the representation of a Distinguished Name and a Relative Distinguished Name after encoding according to the specification in [4], such that

```
<distinguished-name> ::= <name>

<relative-distinguished-name> ::= <name-component>
```

where <name> and <name-component> are as defined in [4].

```
LDAPDN ::= LDAPString

RelativeLDAPDN ::= LDAPString
```

Only Attribute Types can be present in a relative distinguished name component; the options of Attribute Descriptions (next section) MUST NOT be used in specifying distinguished names.

4.1.4. Attribute Type

An AttributeType takes on as its value the textual string associated with that AttributeType in its specification.

```
AttributeType ::= LDAPString
```

Each attribute type has a unique OBJECT IDENTIFIER which has been assigned to it. This identifier may be written as decimal digits with components separated by periods, e.g. "2.5.4.10".

A specification may also assign one or more textual names for an attribute type. These names MUST begin with a letter, and only contain ASCII letters, digit characters and hyphens. They are case insensitive. (These ASCII characters are identical to ISO 10646 characters whose UTF-8 encoding is a single byte between 0x00 and 0x7F.)

If the server has a textual name for an attribute type, it MUST use a textual name for attributes returned in search results. The dotted-decimal OBJECT IDENTIFIER is only used if there is no textual name for an attribute type.

Attribute type textual names are non-unique, as two different specifications (neither in standards track RFCs) may choose the same name.

A server which masters or shadows entries SHOULD list all the attribute types it supports in the subschema entries, using the attributeTypes attribute. Servers which support an open-ended set of attributes SHOULD include at least the attributeTypes value for the 'objectClass' attribute. Clients MAY retrieve the attributeTypes value from subschema entries in order to obtain the OBJECT IDENTIFIER and other information associated with attribute types.

Some attribute type names which are used in this version of LDAP are described in [5]. Servers may implement additional attribute types.

4.1.5. Attribute Description

An AttributeDescription is a superset of the definition of the AttributeType. It has the same ASN.1 definition, but allows additional options to be specified. They are also case insensitive.

```
AttributeDescription ::= LDAPString
```

A value of AttributeDescription is based on the following BNF:

```
<AttributeDescription> ::= <AttributeType> [ ";" <options> ]

<options>  ::= <option> | <option> ";" <options>

<option>   ::= <opt-char> <opt-char>*

<opt-char> ::=  ASCII-equivalent letters, numbers and hyphen
```

Examples of valid AttributeDescription:

```
cn
userCertificate;binary
```

One option, "binary", is defined in this document. Additional options may be defined in IETF standards-track and experimental RFCs. Options beginning with "x-" are reserved for private experiments. Any option could be associated with any AttributeType, although not all combinations may be supported by a server.

An AttributeDescription with one or more options is treated as a subtype of the attribute type without any options. Options present in an AttributeDescription are never mutually exclusive. Implementations MUST generate the <options> list sorted in ascending order, and servers MUST treat any two AttributeDescription with the same AttributeType and options as equivalent. A server will treat an AttributeDescription with any options it does not implement as an unrecognized attribute type.

The data type "AttributeDescriptionList" describes a list of 0 or more attribute types. (A list of zero elements has special significance in the Search request.)

```
AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription
```

4.1.5.1. Binary Option

If the "binary" option is present in an AttributeDescription, it overrides any string-based encoding representation defined for that attribute in [5]. Instead the attribute is to be transferred as a binary value encoded using the Basic Encoding Rules [11]. The syntax of the binary value is an ASN.1 data type definition which is referenced by the "SYNTAX" part of the attribute type definition.

The presence or absence of the "binary" option only affects the transfer of attribute values in protocol; servers store any particular attribute in a single format. If a client requests that a server return an attribute in the binary format, but the server cannot generate that format, the server MUST treat this attribute type as an unrecognized attribute type. Similarly, clients MUST NOT expect servers to return an attribute in binary format if the client requested that attribute by name without the binary option.

This option is intended to be used with attributes whose syntax is a complex ASN.1 data type, and the structure of values of that type is needed by clients. Examples of this kind of syntax are "Certificate" and "CertificateList".

4.1.6. Attribute Value

A field of type AttributeValue takes on as its value either a string encoding of a AttributeValue data type, or an OCTET STRING containing an encoded binary value, depending on whether the "binary" option is present in the companion AttributeDescription to this AttributeValue.

The definition of string encodings for different syntaxes and types may be found in other documents, and in particular [5].

```
AttributeValue ::= OCTET STRING
```

Note that there is no defined limit on the size of this encoding; thus protocol values may include multi-megabyte attributes (e.g. photographs).

Attributes may be defined which have arbitrary and non-printable syntax. Implementations MUST NEITHER simply display nor attempt to decode as ASN.1 a value if its syntax is not known. The implementation may attempt to discover the subschema of the source entry, and retrieve the values of attributeTypes from it.

Clients MUST NOT send attribute values in a request which are not valid according to the syntax defined for the attributes.

4.1.7. Attribute Value Assertion

The AttributeValueAssertion type definition is similar to the one in the X.500 directory standards. It contains an attribute description and a matching rule assertion value suitable for that type.

```
     AttributeValueAssertion ::= SEQUENCE {
             attributeDesc   AttributeDescription,
             assertionValue  AssertionValue }

     AssertionValue ::= OCTET STRING
```

If the "binary" option is present in attributeDesc, this signals to the server that the assertionValue is a binary encoding of the assertion value.

For all the string-valued user attributes described in [5], the assertion value syntax is the same as the value syntax. Clients may use attribute values as assertion values in compare requests and search filters.

Note however that the assertion syntax may be different from the value syntax for other attributes or for non-equality matching rules. These may have an assertion syntax which contains only part of the value. See section 20.2.1.8 of X.501 [6] for examples.

4.1.8. Attribute

An attribute consists of a type and one or more values of that type. (Though attributes MUST have at least one value when stored, due to access control restrictions the set may be empty when transferred in protocol. This is described in section 4.5.2, concerning the PartialAttributeList type.)

```
     Attribute ::= SEQUENCE {
             type    AttributeDescription,
             vals    SET OF AttributeValue }
```

Each attribute value is distinct in the set (no duplicates). The order of attribute values within the vals set is undefined and implementation-dependent, and MUST NOT be relied upon.

4.1.9. Matching Rule Identifier

A matching rule is a means of expressing how a server should compare an AssertionValue received in a search filter with an abstract data value. The matching rule defines the syntax of the assertion value and the process to be performed in the server.

An X.501(1993) Matching Rule is identified in the LDAP protocol by the printable representation of its OBJECT IDENTIFIER, either as one of the strings given in [5], or as decimal digits with components separated by periods, e.g. "caseIgnoreIA5Match" or "1.3.6.1.4.1.453.33.33".

```
     MatchingRuleId ::= LDAPString
```

Servers which support matching rules for use in the extensibleMatch search filter MUST list the matching rules they implement in subschema entries, using the matchingRules attributes. The server SHOULD also list there, using the matchingRuleUse attribute, the attribute types with which each matching rule can be used. More information is given in section 4.4 of [5].

4.1.10. Result Message

The LDAPResult is the construct used in this protocol to return success or failure indications from servers to clients. In response to various requests servers will return responses containing fields of type LDAPResult to indicate the final status of a protocol operation request.

```
     LDAPResult ::= SEQUENCE {
             resultCode      ENUMERATED {
                          success                      (0),
                          operationsError              (1),
                          protocolError                (2),
                          timeLimitExceeded            (3),
                          sizeLimitExceeded            (4),
                          compareFalse                 (5),
                          compareTrue                  (6),

                          authMethodNotSupported       (7),
                          strongAuthRequired           (8),
                                     -- 9 reserved --
                          referral                     (10),  -- new
                          adminLimitExceeded           (11),  -- new
                          unavailableCriticalExtension (12),  -- new
                          confidentialityRequired      (13),  -- new
                          saslBindInProgress           (14),  -- new
                          noSuchAttribute              (16),
                          undefinedAttributeType       (17),
                          inappropriateMatching        (18),
                          constraintViolation          (19),
                          attributeOrValueExists       (20),
                          invalidAttributeSyntax       (21),
                                     -- 22-31 unused --
```

```
                             noSuchObject                 (32),
                             aliasProblem                 (33),
                             invalidDNSyntax              (34),
                             -- 35 reserved for undefined isLeaf --
                             aliasDereferencingProblem    (36),
                                        -- 37-47 unused --
                             inappropriateAuthentication  (48),
                             invalidCredentials           (49),
                             insufficientAccessRights     (50),
                             busy                         (51),
                             unavailable                  (52),
                             unwillingToPerform           (53),
                             loopDetect                   (54),
                                        -- 55-63 unused --
                             namingViolation              (64),
                             objectClassViolation         (65),
                             notAllowedOnNonLeaf          (66),
                             notAllowedOnRDN              (67),
                             entryAlreadyExists           (68),
                             objectClassModsProhibited    (69),
                                        -- 70 reserved for CLDAP --
                             affectsMultipleDSAs          (71), -- new
                                        -- 72-79 unused --
                             other                        (80) },
                             -- 81-90 reserved for APIs --
                matchedDN       LDAPDN,
                errorMessage    LDAPString,
                referral        [3] Referral OPTIONAL }
```

All the result codes with the exception of success, compareFalse and compareTrue are to be treated as meaning the operation could not be completed in its entirety.

Most of the result codes are based on problem indications from X.511 error data types. Result codes from 16 to 21 indicate an AttributeProblem, codes 32, 33, 34 and 36 indicate a NameProblem, codes 48, 49 and 50 indicate a SecurityProblem, codes 51 to 54 indicate a ServiceProblem, and codes 64 to 69 and 71 indicates an UpdateProblem.

If a client receives a result code which is not listed above, it is to be treated as an unknown error condition.

The errorMessage field of this construct may, at the server's option, be used to return a string containing a textual, human-readable (terminal control and page formatting characters should be avoided) error diagnostic. As this error diagnostic is not standardized, implementations MUST NOT rely on the values returned. If the server chooses not to return a textual diagnostic, the errorMessage field of the LDAPResult type MUST contain a zero length string.

For result codes of noSuchObject, aliasProblem, invalidDNSyntax and aliasDereferencingProblem, the matchedDN field is set to the name of the lowest entry (object or alias) in the directory that was matched. If no aliases were dereferenced while attempting to locate the entry, this will be a truncated form of the name provided, or if aliases were dereferenced, of the resulting name, as defined in section 12.5 of X.511 [8]. The matchedDN field is to be set to a zero length string with all other result codes.

4.1.11. Referral

The referral error indicates that the contacted server does not hold the target entry of the request. The referral field is present in an LDAPResult if the LDAPResult.resultCode field value is referral, and absent with all other result codes. It contains a reference to another server (or set of servers) which may be accessed via LDAP or other protocols. Referrals can be returned in response to any operation request (except unbind and abandon which do not have responses). At least one URL MUST be present in the Referral.

The referral is not returned for a singleLevel or wholeSubtree search in which the search scope spans multiple naming contexts, and several different servers would need to be contacted to complete the operation. Instead, continuation references, described in section 4.5.3, are returned.

```
Referral ::= SEQUENCE OF LDAPURL  -- one or more

LDAPURL ::= LDAPString -- limited to characters permitted in URLs
```

If the client wishes to progress the operation, it MUST follow the referral by contacting any one of servers. All the URLs MUST be equally capable of being used to progress the operation. (The mechanisms for how this is achieved by multiple servers are outside the scope of this document.)

URLs for servers implementing the LDAP protocol are written according to [9]. If an alias was dereferenced, the <dn> part of the URL MUST be present, with the new target object name. If the <dn> part is present, the client MUST use this name in its next request to progress the operation, and if it is not present the client will use the same name as in the original request. Some servers (e.g. participating in distributed indexing) may provide a different filter in a referral for a search operation. If the filter part of the URL is present in an LDAPURL, the client MUST use this filter in its next request to progress this search, and if it is not present the client MUST use the same filter as it used for that search. Other aspects of the new request may be the same or different as the request which generated the referral.

Note that UTF-8 characters appearing in a DN or search filter may not be legal for URLs (e.g. spaces) and MUST be escaped using the % method in RFC 1738 [7].

Other kinds of URLs may be returned, so long as the operation could be performed using that protocol.

4.1.12. Controls

A control is a way to specify extension information. Controls which are sent as part of a request apply only to that request and are not saved.

```
     Controls ::= SEQUENCE OF Control

     Control ::= SEQUENCE {
             controlType             LDAPOID,
             criticality             BOOLEAN DEFAULT FALSE,
             controlValue            OCTET STRING OPTIONAL }
```

The controlType field MUST be a UTF-8 encoded dotted-decimal representation of an OBJECT IDENTIFIER which uniquely identifies the control. This prevents conflicts between control names.

The criticality field is either TRUE or FALSE.

If the server recognizes the control type and it is appropriate for the operation, the server will make use of the control when performing the operation.

If the server does not recognize the control type and the criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is not appropriate for the operation and criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unsupportedCriticalExtension.

If the control is unrecognized or inappropriate but the criticality field is FALSE, the server MUST ignore the control.

The controlValue contains any information associated with the control, and its format is defined for the control. The server MUST be prepared to handle arbitrary contents of the controlValue octet string, including zero bytes. It is absent only if there is no value information which is associated with a control of its type.

This document does not define any controls. Controls may be defined in other documents. The definition of a control consists of:

- the OBJECT IDENTIFIER assigned to the control,

- whether the control is always noncritical, always critical, or critical at the client's option,

- the format of the controlValue contents of the control.

Servers list the controls which they recognize in the supportedControl attribute in the root DSE.

4.2. Bind Operation

The function of the Bind Operation is to allow authentication information to be exchanged between the client and server.

The Bind Request is defined as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                 -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

Parameters of the Bind Request are:

- version: A version number indicating the version of the protocol to be used in this protocol session. This document describes version 3 of the LDAP protocol. Note that there is no version negotiation, and the client just sets this parameter to the version it desires. If the client requests protocol version 2, a server that supports the version 2 protocol as described in [2] will not return any v3-

specific protocol fields. (Note that not all LDAP servers will support protocol version 2, since they may be unable to generate the attribute syntaxes associated with version 2.)

- name: The name of the directory object that the client wishes to bind as. This field may take on a null value (a zero length string) for the purposes of anonymous binds, when authentication has been performed at a lower layer, or when using SASL credentials with a mechanism that includes the LDAPDN in the credentials.

- authentication: information used to authenticate the name, if any, provided in the Bind Request.

Upon receipt of a Bind Request, a protocol server will authenticate the requesting client, if necessary. The server will then return a Bind Response to the client indicating the status of the authentication.

Authorization is the use of this authentication information when performing operations. Authorization MAY be affected by factors outside of the LDAP Bind request, such as lower layer security services.

4.2.1. Sequencing of the Bind Request

For some SASL authentication mechanisms, it may be necessary for the client to invoke the BindRequest multiple times. If at any stage the client wishes to abort the bind process it MAY unbind and then drop the underlying connection. Clients MUST NOT invoke operations between two Bind requests made as part of a multi-stage bind.

A client may abort a SASL bind negotiation by sending a BindRequest with a different value in the mechanism field of SaslCredentials, or an AuthenticationChoice other than sasl.

If the client sends a BindRequest with the sasl mechanism field as an empty string, the server MUST return a BindResponse with authMethodNotSupported as the resultCode. This will allow clients to abort a negotiation if it wishes to try again with the same SASL mechanism.

Unlike LDAP v2, the client need not send a Bind Request in the first PDU of the connection. The client may request any operations and the server MUST treat these as unauthenticated. If the server requires that the client bind before browsing or modifying the directory, the server MAY reject a request other than binding, unbinding or an extended request with the "operationsError" result.

If the client did not bind before sending a request and receives an operationsError, it may then send a Bind Request. If this also fails or the client chooses not to bind on the existing connection, it will close the connection, reopen it and begin again by first sending a PDU with a Bind Request. This will aid in interoperating with servers implementing other versions of LDAP.

Clients MAY send multiple bind requests on a connection to change their credentials. A subsequent bind process has the effect of abandoning all operations outstanding on the connection. (This simplifies server implementation.) Authentication from earlier binds are subsequently ignored, and so if the bind fails, the connection will be treated as anonymous. If a SASL transfer encryption or integrity mechanism has been negotiated, and that mechanism does not support the changing of credentials from one identity to another, then the client MUST instead establish a new connection.

4.2.2. Authentication and Other Security Services

The simple authentication option provides minimal authentication facilities, with the contents of the authentication field consisting only of a cleartext password. Note that the use of cleartext passwords is not recommended over open networks when there is no authentication or encryption being performed by a lower layer; see the "Security Considerations" section.

If no authentication is to be performed, then the simple authentication option MUST be chosen, and the password be of zero length. (This is often done by LDAPv2 clients.) Typically the DN is also of zero length.

The sasl choice allows for any mechanism defined for use with SASL [12]. The mechanism field contains the name of the mechanism. The credentials field contains the arbitrary data used for authentication, inside an OCTET STRING wrapper. Note that unlike some Internet application protocols where SASL is used, LDAP is not text-based, thus no base64 transformations are performed on the credentials.

If any SASL-based integrity or confidentiality services are enabled, they take effect following the transmission by the server and reception by the client of the final BindResponse with resultCode success.

The client can request that the server use authentication information from a lower layer protocol by using the SASL EXTERNAL mechanism.

4.2.3. Bind Response

The Bind Response is defined as follows.

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
     COMPONENTS OF LDAPResult,
     serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

BindResponse consists simply of an indication from the server of he status of the client's request for authentication.

f the bind was successful, the resultCode will be success, therwise it will be one of:

- operationsError: server encountered an internal error,

- protocolError: unrecognized version number or incorrect PDU structure,

- authMethodNotSupported: unrecognized SASL mechanism name,

- strongAuthRequired: the server requires authentication be performed with a SASL mechanism,

- referral: this server cannot accept this bind and the client should try another,

- saslBindInProgress: the server requires the client to send a new bind request, with the same sasl mechanism, to continue the authentication process,

- inappropriateAuthentication: the server requires the client which had attempted to bind anonymously or without supplying credentials to provide some form of credentials,

- invalidCredentials: the wrong password was supplied or the SASL credentials could not be processed,

- unavailable: the server is shutting down.

If the server does not support the client's requested protocol version, it MUST set the resultCode to protocolError.

If the client receives a BindResponse response where the resultCode was protocolError, it MUST close the connection as the server will be unwilling to accept further operations. (This is for compatibility with earlier versions of LDAP, in which the bind was always the first operation, and there was no negotiation.)

The serverSaslCreds are used as part of a SASL-defined bind mechanism to allow the client to authenticate the server to which it is communicating, or to perform "challenge-response" authentication. If the client bound with the password choice, or the SASL mechanism does not require the server to return information to the client, then this field is not to be included in the result.

4.3. Unbind Operation

The function of the Unbind Operation is to terminate a protocol session. The Unbind Operation is defined as follows:

```
UnbindRequest ::= [APPLICATION 2] NULL
```

The Unbind Operation has no response defined. Upon transmission of an UnbindRequest, a protocol client may assume that the protocol session is terminated. Upon receipt of an UnbindRequest, a protocol server may assume that the requesting client has terminated the session and that all outstanding requests may be discarded, and may close the connection.

4.4. Unsolicited Notification

An unsolicited notification is an LDAPMessage sent from the server to the client which is not in response to any LDAPMessage received by the server. It is used to signal an extraordinary condition in the server or in the connection between the client and the server. The notification is of an advisory nature, and the server will not expect any response to be returned from the client.

The unsolicited notification is structured as an LDAPMessage in which the messageID is 0 and protocolOp is of the extendedResp form. The responseName field of the ExtendedResponse is present. The LDAPOID value MUST be unique for this notification, and not be used in any other situation.

One unsolicited notification is defined in this document.

4.4.1. Notice of Disconnection

This notification may be used by the server to advise the client that the server is about to close the connection due to an error condition. Note that this notification is NOT a response to an unbind requested by the client: the server MUST follow the procedures of section 4.3. This notification is intended to assist clients in distinguishing between an error condition and a transient network failure. As with a connection close due to network failure, the client MUST NOT assume that any outstanding requests which modified the directory have succeeded or failed.

The responseName is 1.3.6.1.4.1.1466.20036, the response field is absent, and the resultCode is used to indicate the reason for the disconnection.

The following resultCode values are to be used in this notification:

- protocolError: The server has received data from the client in which
  the LDAPMessage structure could not be parsed.

- strongAuthRequired: The server has detected that an established underlying security association protecting communication between the client and server has unexpectedly failed or been compromised.

- unavailable: This server will stop accepting new connections and operations on all existing connections, and be unavailable for an extended period of time. The client may make use of an alternative server.

After sending this notice, the server MUST close the connection. After receiving this notice, the client MUST NOT transmit any further on the connection, and may abruptly close the connection.

4.5. Search Operation

The Search Operation allows a client to request that a search be performed on its behalf by a server. This can be used to read attributes from a single entry, from entries immediately below a particular entry, or a whole subtree of entries.

4.5.1. Search Request

The Search Request is defined as follows:

```
     SearchRequest ::= [APPLICATION 3] SEQUENCE {
             baseObject      LDAPDN,
             scope           ENUMERATED {
                     baseObject              (0),
                     singleLevel             (1),
                     wholeSubtree            (2) },
             derefAliases    ENUMERATED {
                     neverDerefAliases       (0),
                     derefInSearching        (1),
                     derefFindingBaseObj     (2),
```

```
                        derefAlways             (3) },
                sizeLimit       INTEGER (0 .. maxInt),
                timeLimit       INTEGER (0 .. maxInt),
                typesOnly       BOOLEAN,
                filter          Filter,
                attributes      AttributeDescriptionList }

        Filter ::= CHOICE {
                and             [0] SET OF Filter,
                or              [1] SET OF Filter,
                not             [2] Filter,
                equalityMatch   [3] AttributeValueAssertion,
                substrings      [4] SubstringFilter,
                greaterOrEqual  [5] AttributeValueAssertion,
                lessOrEqual     [6] AttributeValueAssertion,
                present         [7] AttributeDescription,
                approxMatch     [8] AttributeValueAssertion,
                extensibleMatch [9] MatchingRuleAssertion }

        SubstringFilter ::= SEQUENCE {
                type            AttributeDescription,
                -- at least one must be present
                substrings      SEQUENCE OF CHOICE {
                        initial [0] LDAPString,
                        any     [1] LDAPString,
                        final   [2] LDAPString } }

        MatchingRuleAssertion ::= SEQUENCE {
                matchingRule    [1] MatchingRuleId OPTIONAL,
                type            [2] AttributeDescription OPTIONAL,
                matchValue      [3] AssertionValue,
                dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

Parameters of the Search Request are:

- baseObject: An LDAPDN that is the base object entry relative to which the search is to be performed.

- scope: An indicator of the scope of the search to be performed. The semantics of the possible values of this field are identical to the semantics of the scope field in the X.511 Search Operation.

- derefAliases: An indicator as to how alias objects (as defined in X.501) are to be handled in searching. The semantics of the possible values of this field are:

  neverDerefAliases: do not dereference aliases in searching or in locating the base object of the search;

derefInSearching: dereference aliases in subordinates of the base object in searching, but not in locating the base object of the search;

derefFindingBaseObj: dereference aliases in locating the base object of the search, but not when searching subordinates of the base object;

derefAlways: dereference aliases both in searching and in locating the base object of the search.

- sizelimit: A sizelimit that restricts the maximum number of entries to be returned as a result of the search. A value of 0 in this field indicates that no client-requested sizelimit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return.

- timelimit: A timelimit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested timelimit restrictions are in effect for the search.

- typesOnly: An indicator as to whether search results will contain both attribute types and values, or just attribute types. Setting this field to TRUE causes only attribute types (no values) to be returned. Setting this field to FALSE causes both attribute types and values to be returned.

- filter: A filter that defines the conditions that must be fulfilled in order for the search to match a given entry.

  The 'and', 'or' and 'not' choices can be used to form combinations of filters. At least one filter element MUST be present in an 'and' or 'or' choice. The others match against individual attribute values of entries in the scope of the search. (Implementor's note: the 'not' filter is an example of a tagged choice in an implicitly-tagged module. In BER this is treated as if the tag was explicit.)

  A server MUST evaluate filters according to the three-valued logic of X.511(93) section 7.8.1. In summary, a filter is evaluated to either "TRUE", "FALSE" or "Undefined". If the filter evaluates to TRUE for a particular entry, then the attributes of that entry are returned as part of the search result (subject to any applicable access control restrictions). If the filter evaluates to FALSE or Undefined, then the entry is ignored for the search.

A filter of the "and" choice is TRUE if all the filters in the SET OF evaluate to TRUE, FALSE if at least one filter is FALSE, and otherwise Undefined. A filter of the "or" choice is FALSE if all of the filters in the SET OF evaluate to FALSE, TRUE if at least one filter is TRUE, and Undefined otherwise. A filter of the "not" choice is TRUE if the filter being negated is FALSE, FALSE if it is TRUE, and Undefined if it is Undefined.

The present match evaluates to TRUE where there is an attribute or subtype of the specified attribute description present in an entry, and FALSE otherwise (including a presence test with an unrecognized attribute description.)

The extensibleMatch is new in this version of LDAP. If the matchingRule field is absent, the type field MUST be present, and the equality match is performed for that type. If the type field is absent and matchingRule is present, the matchValue is compared against all attributes in an entry which support that matchingRule, and the matchingRule determines the syntax for the assertion value (the filter item evaluates to TRUE if it matches with at least one attribute in the entry, FALSE if it does not match any attribute in the entry, and Undefined if the matchingRule is not recognized or the assertionValue cannot be parsed.) If the type field is present and matchingRule is present, the matchingRule MUST be one permitted for use with that type, otherwise the filter item is undefined. If the dnAttributes field is set to TRUE, the match is applied against all the attributes in an entry's distinguished name as well, and also evaluates to TRUE if there is at least one attribute in the distinguished name for which the filter item evaluates to TRUE. (Editors note: The dnAttributes field is present so that there does not need to be multiple versions of generic matching rules such as for word matching, one to apply to entries and another to apply to entries and dn attributes as well).

A filter item evaluates to Undefined when the server would not be able to determine whether the assertion value matches an entry. If an attribute description in an equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch or extensibleMatch filter is not recognized by the server, a matching rule id in the extensibleMatch is not recognized by the server, the assertion value cannot be parsed, or the type of filtering requested is not implemented, then the filter is Undefined. Thus for example if a server did not recognize the attribute type shoeSize, a filter of (shoeSize=*) would evaluate to FALSE, and the filters (shoeSize=12), (shoeSize>=12) and (shoeSize<=12) would evaluate to Undefined.

Servers MUST NOT return errors if attribute descriptions or matching rule ids are not recognized, or assertion values cannot be parsed. More details of filter processing are given in section 7.8 of X.511 [8].

- attributes: A list of the attributes to be returned from each entry which matches the search filter. There are two special values which may be used: an empty list with no attributes, and the attribute description string "*". Both of these signify that all user attributes are to be returned. (The "*" allows the client to request all user attributes in addition to specific operational attributes).

  Attributes MUST be named at most once in the list, and are returned at most once in an entry. If there are attribute descriptions in the list which are not recognized, they are ignored by the server.

  If the client does not want any attributes returned, it can specify a list containing only the attribute with OID "1.1". This OID was chosen arbitrarily and does not correspond to any attribute in use.

  Client implementors should note that even if all user attributes are requested, some attributes of the entry may not be included in search results due to access control or other restrictions. Furthermore, servers will not return operational attributes, such as objectClasses or attributeTypes, unless they are listed by name, since there may be extremely large number of values for certain operational attributes. (A list of operational attributes for use in LDAP is given in [5].)

Note that an X.500 "list"-like operation can be emulated by the client requesting a one-level LDAP search operation with a filter checking for the existence of the objectClass attribute, and that an X.500 "read"-like operation can be emulated by a base object LDAP search operation with the same filter. A server which provides a gateway to X.500 is not required to use the Read or List operations, although it may choose to do so, and if it does must provide the same semantics as the X.500 search operation.

4.5.2. Search Result

The results of the search attempted by the server upon receipt of a Search Request are returned in Search Responses, which are LDAP messages containing either SearchResultEntry, SearchResultReference, ExtendedResponse or SearchResultDone data types.

```
        SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
                objectName      LDAPDN,
```

```
                attributes      PartialAttributeList }

        PartialAttributeList ::= SEQUENCE OF SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }
        -- implementors should note that the PartialAttributeList may
        -- have zero elements (if none of the attributes of that entry
        -- were requested, or could be returned), and that the vals set
        -- may also have zero elements (if types only was requested, or
        -- all values were excluded from the result.)

        SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL
        -- at least one LDAPURL element must be present

        SearchResultDone ::= [APPLICATION 5] LDAPResult
```

Upon receipt of a Search Request, a server will perform the necessary search of the DIT.

If the LDAP session is operating over a connection-oriented transport such as TCP, the server will return to the client a sequence of responses in separate LDAP messages. There may be zero or more responses containing SearchResultEntry, one for each entry found during the search. There may also be zero or more responses containing SearchResultReference, one for each area not explored by this server during the search. The SearchResultEntry and SearchResultReference PDUs may come in any order. Following all the SearchResultReference responses and all SearchResultEntry responses to be returned by the server, the server will return a response containing the SearchResultDone, which contains an indication of success, or detailing any errors that have occurred.

Each entry returned in a SearchResultEntry will contain all attributes, complete with associated values if necessary, as specified in the attributes field of the Search Request. Return of attributes is subject to access control and other administrative policy. Some attributes may be returned in binary format (indicated by the AttributeDescription in the response having the binary option present).

Some attributes may be constructed by the server and appear in a SearchResultEntry attribute list, although they are not stored attributes of an entry. Clients MUST NOT assume that all attributes can be modified, even if permitted by access control.

LDAPMessage responses of the ExtendedResponse form are reserved for returning information associated with a control requested by the client. These may be defined in future versions of this document.

4.5.3. Continuation References in the Search Result

If the server was able to locate the entry referred to by the baseObject but was unable to search all the entries in the scope at and under the baseObject, the server may return one or more SearchResultReference, each containing a reference to another set of servers for continuing the operation. A server MUST NOT return any SearchResultReference if it has not located the baseObject and thus has not searched any entries; in this case it would return a SearchResultDone containing a referral resultCode.

In the absence of indexing information provided to a server from servers holding subordinate naming contexts, SearchResultReference responses are not affected by search filters and are always returned when in scope.

The SearchResultReference is of the same data type as the Referral. URLs for servers implementing the LDAP protocol are written according to [9]. The <dn> part MUST be present in the URL, with the new target object name. The client MUST use this name in its next request. Some servers (e.g. part of a distributed index exchange system) may provide a different filter in the URLs of the SearchResultReference. If the filter part of the URL is present in an LDAP URL, the client MUST use the new filter in its next request to progress the search, and if the filter part is absent the client will use again the same filter. Other aspects of the new search request may be the same or different as the search which generated the continuation references.

Other kinds of URLs may be returned so long as the operation could be performed using that protocol.

The name of an unexplored subtree in a SearchResultReference need not be subordinate to the base object.

In order to complete the search, the client MUST issue a new search operation for each SearchResultReference that is returned. Note that the abandon operation described in section 4.11 applies only to a particular operation sent on a connection between a client and server, and if the client has multiple outstanding search operations to different servers, it MUST abandon each operation individually.

4.5.3.1. Example

For example, suppose the contacted server (hosta) holds the entry "O=MNN,C=WW" and the entry "CN=Manager,O=MNN,C=WW". It knows that either LDAP-capable servers (hostb) or (hostc) hold "OU=People,O=MNN,C=WW" (one is the master and the other server a shadow), and that LDAP-capable server (hostd) holds the subtree "OU=Roles,O=MNN,C=WW". If a subtree search of "O=MNN,C=WW" is requested to the contacted server, it may return the following:

```
SearchResultEntry for O=MNN,C=WW
SearchResultEntry for CN=Manager,O=MNN,C=WW
SearchResultReference {
  ldap://hostb/OU=People,O=MNN,C=WW
  ldap://hostc/OU=People,O=MNN,C=WW
}
SearchResultReference {
  ldap://hostd/OU=Roles,O=MNN,C=WW
}
SearchResultDone (success)
```

Client implementors should note that when following a SearchResultReference, additional SearchResultReference may be generated. Continuing the example, if the client contacted the server (hostb) and issued the search for the subtree "OU=People,O=MNN,C=WW", the server might respond as follows:

```
SearchResultEntry for OU=People,O=MNN,C=WW
SearchResultReference {
 ldap://hoste/OU=Managers,OU=People,O=MNN,C=WW
}
SearchResultReference {
 ldap://hostf/OU=Consultants,OU=People,O=MNN,C=WW
}
SearchResultDone (success)
```

If the contacted server does not hold the base object for the search, then it will return a referral to the client. For example, if the client requests a subtree search of "O=XYZ,C=US" to hosta, the server may return only a SearchResultDone containing a referral.

```
SearchResultDone (referral) {
  ldap://hostg/
}
```

4.6. Modify Operation

The Modify Operation allows a client to request that a modification of an entry be performed on its behalf by a server. The Modify Request is defined as follows:

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
```

```
                        operation       ENUMERATED {
                                                add     (0),
                                                delete  (1),
                                                replace (2) },
                        modification    AttributeTypeAndValues } }

        AttributeTypeAndValues ::= SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }
```

Parameters of the Modify Request are:

- object: The object to be modified. The value of this field contains the DN of the entry to be modified. The server will not perform any alias dereferencing in determining the object to be modified.

- modification: A list of modifications to be performed on the entry. The entire list of entry modifications MUST be performed in the order they are listed, as a single atomic operation. While individual modifications may violate the directory schema, the resulting entry after the entire list of modifications is performed MUST conform to the requirements of the directory schema. The values that may be taken on by the 'operation' field in each modification construct have the following semantics respectively:

  add: add values listed to the given attribute, creating the attribute if necessary;

  delete: delete values listed from the given attribute, removing the entire attribute if no values are listed, or if all current values of the attribute are listed for deletion;

  replace: replace all existing values of the given attribute with the new values listed, creating the attribute if it did not already exist. A replace with no value will delete the entire attribute if it exists, and is ignored if the attribute does not exist.

The result of the modify attempted by the server upon receipt of a Modify Request is returned in a Modify Response, defined as follows:

```
        ModifyResponse ::= [APPLICATION 7] LDAPResult
```

Upon receipt of a Modify Request, a server will perform the necessary modifications to the DIT.

The server will return to the client a single Modify Response indicating either the successful completion of the DIT modification, or the reason that the modification failed. Note that due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications of the DIT have been performed if the Modify Response received indicates any sort of error, and that all requested modifications have been performed if the Modify Response indicates successful completion of the Modify Operation. If the connection fails, whether the modification occurred or not is indeterminate.

The Modify Operation cannot be used to remove from an entry any of its distinguished values, those values which form the entry's relative distinguished name. An attempt to do so will result in the server returning the error notAllowedOnRDN. The Modify DN Operation described in section 4.9 is used to rename an entry.

If an equality match filter has not been defined for an attribute type, clients MUST NOT attempt to delete individual values of that attribute from an entry using the "delete" form of a modification, and MUST instead use the "replace" form.

Note that due to the simplifications made in LDAP, there is not a direct mapping of the modifications in an LDAP ModifyRequest onto the EntryModifications of a DAP ModifyEntry operation, and different implementations of LDAP-DAP gateways may use different means of representing the change. If successful, the final effect of the operations on the entry MUST be identical.

4.7. Add Operation

The Add Operation allows a client to request the addition of an entry into the directory. The Add Request is defined as follows:

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

Parameters of the Add Request are:

- entry: the Distinguished Name of the entry to be added. Note that the server will not dereference any aliases in locating the entry to be added.

- attributes: the list of attributes that make up the content of the entry being added. Clients MUST include distinguished values (those forming the entry's own RDN) in this list, the objectClass attribute, and values of any mandatory attributes of the listed object classes. Clients MUST NOT supply the createTimestamp or creatorsName attributes, since these will be generated automatically by the server.

The entry named in the entry field of the AddRequest MUST NOT exist for the AddRequest to succeed. The parent of the entry to be added MUST exist. For example, if the client attempted to add "CN=JS,O=Foo,C=US", the "O=Foo,C=US" entry did not exist, and the "C=US" entry did exist, then the server would return the error noSuchObject with the matchedDN field containing "C=US". If the parent entry exists but is not in a naming context held by the server, the server SHOULD return a referral to the server holding the parent entry.

Servers implementations SHOULD NOT restrict where entries can be located in the directory. Some servers MAY allow the administrator to restrict the classes of entries which can be added to the directory.

Upon receipt of an Add Request, a server will attempt to perform the add requested. The result of the add attempt will be returned to the client in the Add Response, defined as follows:

```
AddResponse ::= [APPLICATION 9] LDAPResult
```

A response of success indicates that the new entry is present in the directory.

4.8. Delete Operation

The Delete Operation allows a client to request the removal of an entry from the directory. The Delete Request is defined as follows:

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

The Delete Request consists of the Distinguished Name of the entry to be deleted. Note that the server will not dereference aliases while resolving the name of the target entry to be removed, and that only leaf entries (those with no subordinate entries) can be deleted with this operation.

The result of the delete attempted by the server upon receipt of a Delete Request is returned in the Delete Response, defined as follows:

```
DelResponse ::= [APPLICATION 11] LDAPResult
```

Upon receipt of a Delete Request, a server will attempt to perform the entry removal requested. The result of the delete attempt will be returned to the client in the Delete Response.

4.9. Modify DN Operation

The Modify DN Operation allows a client to change the leftmost (least significant) component of the name of an entry in the directory, or to move a subtree of entries to a new location in the directory. The Modify DN Request is defined as follows:

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

Parameters of the Modify DN Request are:

- entry: the Distinguished Name of the entry to be changed. This entry may or may not have subordinate entries.

- newrdn: the RDN that will form the leftmost component of the new name of the entry.

- deleteoldrdn: a boolean parameter that controls whether the old RDN attribute values are to be retained as attributes of the entry, or deleted from the entry.

- newSuperior: if present, this is the Distinguished Name of the entry which becomes the immediate superior of the existing entry.

The result of the name change attempted by the server upon receipt of a Modify DN Request is returned in the Modify DN Response, defined as follows:

```
ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

Upon receipt of a ModifyDNRequest, a server will attempt to perform the name change. The result of the name change attempt will be returned to the client in the Modify DN Response.

For example, if the entry named in the "entry" parameter was "cn=John Smith,c=US", the newrdn parameter was "cn=John Cougar Smith", and the newSuperior parameter was absent, then this operation would attempt to rename the entry to be "cn=John Cougar Smith,c=US". If there was already an entry with that name, the operation would fail with error code entryAlreadyExists.

If the deleteoldrdn parameter is TRUE, the values forming the old RDN are deleted from the entry. If the deleteoldrdn parameter is FALSE, the values forming the old RDN will be retained as non-distinguished attribute values of the entry. The server may not perform the operation and return an error code if the setting of the deleteoldrdn parameter would cause a schema inconsistency in the entry.

Note that X.500 restricts the ModifyDN operation to only affect entries that are contained within a single server. If the LDAP server is mapped onto DAP, then this restriction will apply, and the resultCode affectsMultipleDSAs will be returned if this error occurred. In general clients MUST NOT expect to be able to perform arbitrary movements of entries and subtrees between servers.

4.10. Compare Operation

The Compare Operation allows a client to compare an assertion provided with an entry in the directory. The Compare Request is defined as follows:

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }
```

Parameters of the Compare Request are:

- entry: the name of the entry to be compared with.

- ava: the assertion with which an attribute in the entry is to be compared.

The result of the compare attempted by the server upon receipt of a Compare Request is returned in the Compare Response, defined as follows:

```
CompareResponse ::= [APPLICATION 15] LDAPResult
```

Upon receipt of a Compare Request, a server will attempt to perform the requested comparison. The result of the comparison will be returned to the client in the Compare Response. Note that errors and the result of comparison are all returned in the same construct.

Note that some directory systems may establish access controls which permit the values of certain attributes (such as userPassword) to be compared but not read. In a search result, it may be that an attribute of that type would be returned, but with an empty set of values.

4.11. Abandon Operation

The function of the Abandon Operation is to allow a client to request that the server abandon an outstanding operation. The Abandon Request is defined as follows:

```
AbandonRequest ::= [APPLICATION 16] MessageID
```

The MessageID MUST be that of a an operation which was requested earlier in this connection.

(The abandon request itself has its own message id. This is distinct from the id of the earlier operation being abandoned.)

There is no response defined in the Abandon Operation. Upon transmission of an Abandon Operation, a client may expect that the operation identified by the Message ID in the Abandon Request has been abandoned. In the event that a server receives an Abandon Request on a Search Operation in the midst of transmitting responses to the search, that server MUST cease transmitting entry responses to the abandoned request immediately, and MUST NOT send the SearchResponseDone. Of course, the server MUST ensure that only properly encoded LDAPMessage PDUs are transmitted.

Clients MUST NOT send abandon requests for the same operation multiple times, and MUST also be prepared to receive results from operations it has abandoned (since these may have been in transit when the abandon was requested).

Servers MUST discard abandon requests for message IDs they do not recognize, for operations which cannot be abandoned, and for operations which have already been abandoned.

4.12. Extended Operation

An extension mechanism has been added in this version of LDAP, in order to allow additional operations to be defined for services not available elsewhere in this protocol, for instance digitally signed operations and results.

The extended operation allows clients to make requests and receive responses with predefined syntaxes and semantics. These may be defined in RFCs or be private to particular implementations. Each request MUST have a unique OBJECT IDENTIFIER assigned to it.

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

The requestName is a dotted-decimal representation of the OBJECT IDENTIFIER corresponding to the request. The requestValue is information in a form defined by that request, encapsulated inside an OCTET STRING.

The server will respond to this with an LDAPMessage containing the ExtendedResponse.

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

If the server does not recognize the request name, it MUST return only the response fields from LDAPResult, containing the protocolError result code.

5. Protocol Element Encodings and Transfer

One underlying service is defined here. Clients and servers SHOULD implement the mapping of LDAP over TCP described in 5.2.1.

5.1. Mapping Onto BER-based Transport Services

The protocol elements of LDAP are encoded for exchange using the Basic Encoding Rules (BER) [11] of ASN.1 [3]. However, due to the high overhead involved in using certain elements of the BER, the following additional restrictions are placed on BER-encodings of LDAP protocol elements:

(1) Only the definite form of length encoding will be used.

(2) OCTET STRING values will be encoded in the primitive form only.

(3) If the value of a BOOLEAN type is true, the encoding MUST have its contents octets set to hex "FF".

(4) If a value of a type is its default value, it MUST be absent. Only some BOOLEAN and INTEGER types have default values in this protocol definition.

These restrictions do not apply to ASN.1 types encapsulated inside of OCTET STRING values, such as attribute values, unless otherwise noted.

5.2. Transfer Protocols

This protocol is designed to run over connection-oriented, reliable transports, with all 8 bits in an octet being significant in the data stream.

5.2.1. Transmission Control Protocol (TCP)

The LDAPMessage PDUs are mapped directly onto the TCP bytestream. It is recommended that server implementations running over the TCP MAY provide a protocol listener on the assigned port, 389. Servers may instead provide a listener on a different port number. Clients MUST support contacting servers on any valid TCP port.

6. Implementation Guidelines

This document describes an Internet protocol.

6.1. Server Implementations

The server MUST be capable of recognizing all the mandatory attribute type names and implement the syntaxes specified in [5]. Servers MAY also recognize additional attribute type names.

6.2. Client Implementations

Clients which request referrals MUST ensure that they do not loop between servers. They MUST NOT repeatedly contact the same server for the same request with the same target entry name, scope and filter. Some clients may be using a counter that is incremented each time referral handling occurs for an operation, and these kinds of clients MUST be able to handle a DIT with at least ten layers of naming contexts between the root and a leaf entry.

In the absence of prior agreements with servers, clients SHOULD NOT assume that servers support any particular schemas beyond those referenced in section 6.1. Different schemas can have different attribute types with the same names. The client can retrieve the subschema entries referenced by the subschemaSubentry attribute in the server's root DSE or in entries held by the server.

7. Security Considerations

When used with a connection-oriented transport, this version of the protocol provides facilities for the LDAP v2 authentication mechanism, simple authentication using a cleartext password, as well as any SASL mechanism [12]. SASL allows for integrity and privacy services to be negotiated.

It is also permitted that the server can return its credentials to the client, if it chooses to do so.

Use of cleartext password is strongly discouraged where the underlying transport service cannot guarantee confidentiality and may result in disclosure of the password to unauthorized parties.

When used with SASL, it should be noted that the name field of the BindRequest is not protected against modification. Thus if the distinguished name of the client (an LDAPDN) is agreed through the negotiation of the credentials, it takes precedence over any value in the unprotected name field.

Implementations which cache attributes and entries obtained via LDAP MUST ensure that access controls are maintained if that information is to be provided to multiple clients, since servers may have access control policies which prevent the return of entries or attributes in search results except to particular authenticated clients. For example, caches could serve result information only to the client whose request caused it to be cache.

8. Acknowledgements

This document is an update to RFC 1777, by Wengyik Yeong, Tim Howes, and Steve Kille. Design ideas included in this document are based on those discussed in ASID and other IETF Working Groups. The contributions of individuals in these working groups is gratefully acknowledged.

9. Bibliography

[1] ITU-T Rec. X.500, "The Directory: Overview of Concepts, Models and Service", 1993.

[2] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol", RFC 1777, March 1995.

[3] ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994.

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

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

[6] ITU-T Rec. X.501, "The Directory: Models", 1993.

[7] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[8] ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993.

[9] Howes, T., and M. Smith, "The LDAP URL Format", RFC 2255, December 1997.

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

[11] ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994.

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

[13] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

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

10. Authors' Addresses

Mark Wahl
Critical Angle Inc.
4815 W Braker Lane #502-385
Austin, TX 78759
USA

Phone: +1 512 372-3160
EMail: M.Wahl@critical-angle.com

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd., MS MV068
Mountain View, CA 94043
USA

Phone: +1 650 937-3419
EMail: howes@netscape.com

Steve Kille
Isode Limited
The Dome, The Square
Richmond
TW9 1DT
UK

Phone: +44-181-332-9091
EMail: S.Kille@isode.com

Appendix A - Complete ASN.1 Definition

```
        Lightweight-Directory-Access-Protocol-V3 DEFINITIONS
        IMPLICIT TAGS ::=

        BEGIN

        LDAPMessage ::= SEQUENCE {
                messageID       MessageID,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindResponse    BindResponse,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
                        abandonRequest  AbandonRequest,
                        extendedReq     ExtendedRequest,
                        extendedResp    ExtendedResponse },
                 controls       [0] Controls OPTIONAL }

        MessageID ::= INTEGER (0 .. maxInt)

        maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

        LDAPString ::= OCTET STRING

        LDAPOID ::= OCTET STRING

        LDAPDN ::= LDAPString

        RelativeLDAPDN ::= LDAPString

        AttributeType ::= LDAPString

        AttributeDescription ::= LDAPString
```

```
        AttributeDescriptionList ::= SEQUENCE OF
                AttributeDescription

        AttributeValue ::= OCTET STRING

        AttributeValueAssertion ::= SEQUENCE {
                attributeDesc   AttributeDescription,
                assertionValue  AssertionValue }

        AssertionValue ::= OCTET STRING

        Attribute ::= SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }

        MatchingRuleId ::= LDAPString

        LDAPResult ::= SEQUENCE {
                resultCode      ENUMERATED {
                             success                      (0),
                             operationsError              (1),
                             protocolError                (2),
                             timeLimitExceeded            (3),
                             sizeLimitExceeded            (4),
                             compareFalse                 (5),
                             compareTrue                  (6),
                             authMethodNotSupported       (7),
                             strongAuthRequired           (8),
                                        -- 9 reserved --
                             referral                     (10),  -- new
                             adminLimitExceeded           (11),  -- new
                             unavailableCriticalExtension (12),  -- new
                             confidentialityRequired      (13),  -- new
                             saslBindInProgress           (14),  -- new
                             noSuchAttribute              (16),
                             undefinedAttributeType       (17),
                             inappropriateMatching        (18),
                             constraintViolation          (19),
                             attributeOrValueExists       (20),
                             invalidAttributeSyntax       (21),
                                        -- 22-31 unused --
                             noSuchObject                 (32),
                             aliasProblem                 (33),
                             invalidDNSyntax              (34),
                             -- 35 reserved for undefined isLeaf --
                             aliasDereferencingProblem    (36),
                                        -- 37-47 unused --
                             inappropriateAuthentication  (48),
```

```
                             invalidCredentials           (49),
                             insufficientAccessRights     (50),
                             busy                         (51),
                             unavailable                  (52),
                             unwillingToPerform           (53),
                             loopDetect                   (54),
                                        -- 55-63 unused --
                             namingViolation              (64),
                             objectClassViolation         (65),
                             notAllowedOnNonLeaf          (66),
                             notAllowedOnRDN              (67),
                             entryAlreadyExists           (68),
                             objectClassModsProhibited    (69),
                                        -- 70 reserved for CLDAP --
                             affectsMultipleDSAs          (71), -- new
                                        -- 72-79 unused --
                             other                        (80) },
                             -- 81-90 reserved for APIs --
                matchedDN       LDAPDN,
                errorMessage    LDAPString,
                referral        [3] Referral OPTIONAL }

        Referral ::= SEQUENCE OF LDAPURL

        LDAPURL ::= LDAPString -- limited to characters permitted in URLs

        Controls ::= SEQUENCE OF Control

        Control ::= SEQUENCE {
                controlType             LDAPOID,
                criticality             BOOLEAN DEFAULT FALSE,
                controlValue            OCTET STRING OPTIONAL }

        BindRequest ::= [APPLICATION 0] SEQUENCE {
                version                 INTEGER (1 .. 127),
                name                    LDAPDN,
                authentication          AuthenticationChoice }

        AuthenticationChoice ::= CHOICE {
                simple                  [0] OCTET STRING,
                                         -- 1 and 2 reserved
                sasl                    [3] SaslCredentials }

        SaslCredentials ::= SEQUENCE {
                mechanism               LDAPString,
                credentials             OCTET STRING OPTIONAL }

        BindResponse ::= [APPLICATION 1] SEQUENCE {
```

```
             COMPONENTS OF LDAPResult,
             serverSaslCreds    [7] OCTET STRING OPTIONAL }

        UnbindRequest ::= [APPLICATION 2] NULL

        SearchRequest ::= [APPLICATION 3] SEQUENCE {
                baseObject      LDAPDN,
                scope           ENUMERATED {
                        baseObject              (0),
                        singleLevel             (1),
                        wholeSubtree            (2) },
                derefAliases    ENUMERATED {
                        neverDerefAliases       (0),
                        derefInSearching        (1),
                        derefFindingBaseObj     (2),
                        derefAlways             (3) },
                sizeLimit       INTEGER (0 .. maxInt),
                timeLimit       INTEGER (0 .. maxInt),
                typesOnly       BOOLEAN,
                filter          Filter,
                attributes      AttributeDescriptionList }

        Filter ::= CHOICE {
                and             [0] SET OF Filter,
                or              [1] SET OF Filter,
                not             [2] Filter,
                equalityMatch   [3] AttributeValueAssertion,
                substrings      [4] SubstringFilter,
                greaterOrEqual  [5] AttributeValueAssertion,
                lessOrEqual     [6] AttributeValueAssertion,
                present         [7] AttributeDescription,
                approxMatch     [8] AttributeValueAssertion,
                extensibleMatch [9] MatchingRuleAssertion }

        SubstringFilter ::= SEQUENCE {
                type            AttributeDescription,
                -- at least one must be present
                substrings      SEQUENCE OF CHOICE {
                        initial [0] LDAPString,
                        any     [1] LDAPString,
                        final   [2] LDAPString } }

        MatchingRuleAssertion ::= SEQUENCE {
                matchingRule    [1] MatchingRuleId OPTIONAL,
                type            [2] AttributeDescription OPTIONAL,
                matchValue      [3] AssertionValue,
                dnAttributes    [4] BOOLEAN DEFAULT FALSE }
```

```
        SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
                objectName      LDAPDN,
                attributes      PartialAttributeList }

        PartialAttributeList ::= SEQUENCE OF SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }

        SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

        SearchResultDone ::= [APPLICATION 5] LDAPResult

        ModifyRequest ::= [APPLICATION 6] SEQUENCE {
                object          LDAPDN,
                modification    SEQUENCE OF SEQUENCE {
                        operation       ENUMERATED {
                                                add     (0),
                                                delete  (1),
                                                replace (2) },
                        modification    AttributeTypeAndValues } }

        AttributeTypeAndValues ::= SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }

        ModifyResponse ::= [APPLICATION 7] LDAPResult

        AddRequest ::= [APPLICATION 8] SEQUENCE {
                entry           LDAPDN,
                attributes      AttributeList }

        AttributeList ::= SEQUENCE OF SEQUENCE {
                type    AttributeDescription,
                vals    SET OF AttributeValue }

        AddResponse ::= [APPLICATION 9] LDAPResult

        DelRequest ::= [APPLICATION 10] LDAPDN

        DelResponse ::= [APPLICATION 11] LDAPResult

        ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
                entry           LDAPDN,
                newrdn          RelativeLDAPDN,
                deleteoldrdn    BOOLEAN,
                newSuperior     [0] LDAPDN OPTIONAL }

        ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

```
        CompareRequest ::= [APPLICATION 14] SEQUENCE {
                entry           LDAPDN,
                ava             AttributeValueAssertion }

        CompareResponse ::= [APPLICATION 15] LDAPResult

        AbandonRequest ::= [APPLICATION 16] MessageID

        ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
                requestName      [0] LDAPOID,
                requestValue     [1] OCTET STRING OPTIONAL }

        ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
                COMPONENTS OF LDAPResult,
                responseName     [10] LDAPOID OPTIONAL,
                response         [11] OCTET STRING OPTIONAL }

        END
```