Internet-Draft   Editor: J. Sermersheim
Intended Category: Standard Track   Novell, Inc
Document: draft-ietf-ldapbis-protocol-09.txt   Oct 2002
Obsoletes: RFC 2251

LDAP: The Protocol

Status of this Memo

Distribution of this memo is unlimited. Technical discussion of this document will take place on the IETF LDAP Revision Working Group (LDAPbis) mailing list <ietf-ldapbis@openldap.org>. Please send editorial comments directly to the editor <jimse@novell.com>.

Abstract

This document describes the protocol elements, along with their semantics and encodings, for the Lightweight Directory Access Protocol (LDAP). LDAP provides access to distributed directory services that act in accordance with X.500 data and service models. These protocol elements are based on those described in the X.500 Directory Access Protocol (DAP).

Table of Contents

1. Introduction

The Directory is "a collection of open systems cooperating to provide directory services" [X.500]. A Directory user, which may be a human or other entity, accesses the Directory through a client (or Directory User Agent (DUA)). The client, on behalf of the directory user, interacts with one or more servers (or Directory System Agents

(DSA)). Clients interact with servers using a directory access protocol.

This document details the protocol elements of Lightweight Directory Access Protocol, along with their semantic meanings. Following the description of protocol elements, it describes the way in which the protocol is encoded and transferred.

This document is an integral part of the LDAP Technical Specification [Roadmap].

This document replaces RFC 2251. Appendix C holds a detailed log of changes to RFC 2251. At publication time, this appendix will be distilled to a summary of changes to RFC 2251.

2. Conventions

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 [RFC2119].

3. 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.

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.

Note that the core protocol operations defined in this document can be mapped to a subset of the X.500(1997) directory abstract service. 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.

4. Elements of Protocol

The LDAP protocol is described using Abstract Syntax Notation 1 (ASN.1) [X.680], and is transferred using a subset of ASN.1 Basic Encoding Rules [X.690]. Section 5.1 specifies how the protocol is encoded and transferred.

In order to support future Standards Track extensions to this protocol, extensibility is implied where it is allowed (per ASN.1). In addition, ellipses (...) have been supplied in ASN.1 types that are explicitly extensible as discussed in [LDAPIANA]. Because of the implied extensibility, clients and servers MUST ignore trailing SEQUENCE elements whose tags they do not recognize.

Changes to the LDAP protocol other than those described in [LDAPIANA] require a different version number. A client indicates the version it is using as part of the bind request, described in section 4.2. If a client has not sent a bind, the server MUST assume the client is using version 3 or later.

Comment: Is this true?

Clients may determine the protocol versions a server supports by reading the supportedLDAPVersion attribute from the root DSE [Models]. Servers which implement version 3 or later MUST 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 MAY return the Notice of Disconnection described in section 4.4.1, with resultCode protocolError, and MUST immediately close the connection.

In other cases where the client or server cannot parse a PDU, it SHOULD abruptly close the connection where further communication (including providing notice) would be pernicious. Otherwise, server implementations MUST return an appropriate response to the request, with the resultCode set to protocolError.

The ASN.1 type Controls is defined in section 4.1.11.

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 non-zero value different from the values of any other requests outstanding in the LDAP session of which this message is a part. The zero value is reserved for the unsolicited notification message.

Typical clients increment a counter for each request.

A client MUST NOT send a request with the same message ID as an earlier request on the same connection unless it can be determined that the server is no longer servicing the earlier request. Otherwise the behavior is undefined. For operations that do not return responses (unbind, abandon, and abandoned operations), the client SHOULD assumes the operation is in progress until a subsequent bind request completes.

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

[ISO10646] character set (a superset of Unicode) is used, encoded following the UTF-8 algorithm [RFC2044]. 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 -- UTF-8 encoded,
                                -- ISO 10646 characters
```

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. Although an LDAPOID is encoded as an OCTET STRING, values are limited to the definition of numericoid given in Section 1.3 of [Models].

```
    LDAPOID ::= OCTET STRING -- Constrained to numericoid [Models]
```

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 [LDAPDN].

```
    LDAPDN ::= LDAPString
                -- Constrained to distinguishedName [LDAPDN]

    RelativeLDAPDN ::= LDAPString
                        -- Constrained to name-component [LDAPDN]
```

4.1.4. Attribute Descriptions

The definition and encoding rules for attribute descriptions are defined in Section 2.5 of [Models]. Briefly, an attribute description is an attribute type and zero or more options.

```
    AttributeDescription ::= LDAPString
                              -- Constrained to attributedescription
                              -- [Models]
```

Not all options can be associated with attributes held in the directory. A server will treat an AttributeDescription with any options it does not implement or support as unrecognized. The order in which options appear in the list MUST NOT be used to impart any semantic meaning. Servers MUST treat any two AttributeDescription with the same attribute type and options as equivalent.

AttributeDescriptionList describes a list of 0 or more attribute descriptions. (A list of zero elements has special significance in the Search request.)

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

4.1.5. Attribute Value

A field of type AttributeValue is an OCTET STRING containing an encoded attribute value data type. The value is encoded according to its LDAP-specific encoding definition. The LDAP-specific encoding definitions for different syntaxes and attribute types may be found in other documents, and in particular [Syntaxes].

```
    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 NOT 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 that are not valid according to the syntax defined for the attributes.

4.1.6. 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
```

The syntax of the AssertionValue depends on the context of the LDAP operation being performed. For example, the syntax of the EQUALITY matching rule for an attribute is used when performing a Compare operation. Often this is the same syntax used for values of the attribute type, but in some cases the assertion syntax differs from the value syntax. See objectIdentiferFirstComponentMatch in [Syntaxes] for an example.

4.1.7. Attribute

An attribute consists of an attribute description and one or more values of that attribute description. (Though attributes MUST have at least one value when stored, due to access control restrictions the set may be empty when transferred from the server to the client. 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 set of attribute values is unordered. Implementations MUST NOT reply upon any apparent ordering being repeatable.

4.1.8. Matching Rule Identifier

Matching rules are defined in 4.1.3 of [Models]. A matching rule is identified in the LDAP protocol by the printable representation of either its numericoid, or one of its short name descriptors, 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 4.5 of [Syntaxes].

Comment: move to models

4.1.9. Result Message

The LDAPResult is the construct used in this protocol to return success or failure indications from servers to clients. To various requests, servers will return responses of LDAPResult or responses containing the components of LDAPResponse 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),
                  adminLimitExceeded           (11),
                  unavailableCriticalExtension (12),
                  confidentialityRequired      (13),
                  saslBindInProgress           (14),
                  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),
                                -- 72-79 unused --
                  other                        (80),
                  ... },
                  -- 81-90 reserved for APIs --
           matchedDN       LDAPDN,
           errorMessage    LDAPString,
           referral        [3] Referral OPTIONAL }
```

The result codes enumeration is extensible as defined in Section 3.5 of [LDAPIANA]. The meanings of the result codes are given in Appendix A.

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]. The matchedDN field contains a zero length string with all other result codes.

4.1.10. Referral

The referral result code 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 one or more references to one or more servers or services that 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.

During a search operation, after the baseObject is located, and entries are being evaluated, the referral is not returned. Instead, continuation references, described in section 4.5.3, are returned when the search scope spans multiple naming contexts, and several different servers would need to be contacted to complete the operation.

```
    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 one of the servers. If multiple URLs are present, the client assumes that any URL may be used to progress the operation.

URLs for servers implementing the LDAP protocol are written according to [LDAPDN]. 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 [RFC2396].

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

4.1.11. Controls

A control is a way to specify extension information for an LDAP message. A control only alters the semantics of the message it is attached to.

```
    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 and only applies to request messages that have a corresponding response message. For all other messages (such as abandonRequest, unbindRequest and all response messages), the criticality field is treated as 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 or it is not appropriate for the operation, and the criticality field is TRUE, the server MUST NOT perform the operation, and MUST instead return the resultCode unavailableCriticalExtension.

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. Implementations 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 specify any controls. Controls may be specified in other documents. The specification 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,

- the semantics of the control,

- and optionally, semantics regarding the combination of the control with other controls.

Servers list the controlType of all controls they recognize in the supportedControl attribute [Models] in the root DSE.

Controls should not be combined unless the semantics of the combination has been specified. The semantics of control combinations, if specified, are generally found in the control specification most recently published. In the absence of combination semantics, the behavior of the operation is undefined. Additionally, the order of a combination of controls in the SEQUENCE is ignored unless the control specification(s) describe(s) combination semantics.

4.2. Bind Operation

The function of the Bind Operation is to allow authentication information to be exchanged between the client and server. Prior to the BindRequest, the implied identity is anonymous. Refer to [AuthMeth] for the authentication-related semantics of this operation.

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 server does not support the specified version, it responds with protocolError in the resultCode field of the BindResponse.

- 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 name in the credentials. Server behavior is undefined when the name is a null value, simple authentication is used, and a password is specified. The server SHOULD NOT perform any alias dereferencing in determining the object to bind as.

- authentication: information used to authenticate the name, if any, provided in the Bind Request. This type is extensible as defined in Section 3.6 of [LDAPIANA]. Servers that do not support a choice supplied by a client will return authMethodNotSupported in the result code of the BindResponse.

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.

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.

4.2.2. 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 the status of the client's request for authentication.

If the bind was successful, the resultCode will be success, otherwise

Comment: Reconcile with result cod

e

s

it MAY be one of:

d
r
a
f
t
.

- 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 simple 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 MUST assume that the protocol session is terminated. Upon receipt of an UnbindRequest, a protocol server MUST assume that the requesting client has terminated the session and that all outstanding requests may be discarded, and MUST 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 (Notice of Disconnection) 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.

Comment: move to result code appendix?

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 SIZE (1..MAX) OF Filter,
```

```
        or              [1] SET SIZE (1..MAX) 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,
           -- initial and final can occur at most once
             substrings      SEQUENCE OF CHOICE {
                     initial [0] AssertionValue,
                     any     [1] AssertionValue,
                     final   [2] AssertionValue } }

     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

Comment: When scope is base

or

object of the search;

subtree, the base object is both located

,

and searched. "subordinates" prevents

derefFindingBaseObj: dereference aliases in locating the this o bject from being dereferenced while searching.

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 size limit 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 size limit restrictions are in effect for the search. Servers may enforce a maximum number of entries to return.

- timeLimit: A time limit that restricts the maximum time (in seconds) allowed for a search. A value of 0 in this field indicates that no client-requested time limit 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 (1993) 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 matching rule and assertion syntax for equalityMatch filter items is defined by the EQUALITY matching rule for the attribute type.

The matching rule and assertion syntax for AssertionValues in a substrings filter item is defined by the SUBSTR matching rule for the attribute type.

The matching rule and assertion syntax for greaterOrEqual and lessOrEqual filter items is defined by the ORDERING matching rule for the attribute type.

The matching rule and assertion syntax for approxMatch filter items is implementation-defined. If approximate matching is not supported by the server, the filter item should be treated as an equalityMatch.

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].

- 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 any specified 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 controls 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 [Syntaxes].)

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

If the serverÆs schema defines a textual name for an attribute type, it MUST use a textual name for attributes of that attribute type by specifying one of the textual names as the value of the attribute type. Otherwise, the server uses the object identifier for the attribute type by specifying the object identifier, in ldapOID form, as the value of attribute type.

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 entries, 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 [LDAPDN]. 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. If the originating search scope was singleLevel, the scope part of the URL will be baseObject. 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.

Comment: why not? Probably

because

The name of an unexplored subtree in a SearchResultReference need not

of alias dereferenc

n
c
i
n
g

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, it MUST abandon each operation individually.

4.5.3.1. Example

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

```
SearchResultEntry for DC=Example,DC=NET
SearchResultEntry for CN=Manager,DC=Example,DC=NET
SearchResultReference {
  ldap://hostb/OU=People,DC=Example,DC=NET
  ldap://hostc/OU=People,DC=Example,DC=NET
}
SearchResultReference {
  ldap://hostd/OU=Roles,DC=Example,DC=NET
}
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,DC=Example,DC=NET", the server might respond as follows:

```
SearchResultEntry for OU=People,DC=Example,DC=NET
SearchResultReference {
  ldap://hoste/OU=Managers,OU=People,DC=Example,DC=NET
}
```

```
  SearchResultReference {
    ldap://hostf/OU=Consultants,OU=People,DC=Example,DC=NET
  }
  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 "DC=Example,DC=ORG" 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

Comment: Does not exist on the entry,

attribute does not exist.

the object class, or the schema?

Apply

same to delete

The result of the modification 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 matching rule has not been defined for an attribute type, clients MUST NOT attempt to add or delete individual values of that attribute from an entry using the "add" or "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 NO-USER-MODIFICATION attributes such as the createTimestamp or creatorsName attributes, since the server maintains these automatically.

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,DC=Example,DC=NET", the "DC=Example,DC=NET" entry did not exist, and the "DC=NET" entry did exist, then the server would return the error noSuchObject with the matchedDN field containing "DC=NET". 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 unless DIT structure rules are in place. 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. Note that the server will not dereference any aliases in locating the entry to be changed.

- 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. Note that the server SHOULD NOT dereference any aliases in locating 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
```

Comment: should we talk about what to dowh

Upon receipt of a Compare Request, a server will attempt to perform the requested comparison using the EQUALITY matching rule for the attribute type. The result of the comparison will be returned to the client in the Compare Response. Note that errors and the result of en teh attr ismissing vs

attr isn't

comparison are all returned in the same construct.

in sch

ema?

Should we describe what happens if there's no equal ity matching rule?

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.

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 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, the server MAY abandon the operation identified by the Message ID in the Abandon Request. Operation responses are not sent for successfully abandoned operations. Clients can determine that an operation has been abandoned by performing a subsequent bind operation.

Abandon and Unbind operations cannot be abandoned. The ability to abandon other (particularly update) operations is at the discretion of the server.

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, or are not able to be abandoned).

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. Protocol Encoding

The protocol elements of LDAP are encoded for exchange using the Basic Encoding Rules (BER) [X.690] of ASN.1 [X.680]. 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

Comment: What about control

v

alues

noted.

and extension values?

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 encoded LDAPMessage PDUs are mapped directly onto the TCP bytestream using the BER-based encoding described in section 5.1. It is recommended that server implementations running over the TCP 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 [Syntaxes]. 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 simple authentication using a cleartext password, as well as any SASL mechanism [RFC2222]. 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 in the cache.

8. Acknowledgements

This document is an update to RFC 2251, by Mark Wahl, Tim Howes, and Steve Kille. Their work along with the input of individuals of the IETF LDAPEXT, LDUP, LDAPBIS, and other Working Groups is gratefully acknowledged.

9. Normative References

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

[Roadmap] K. Zeilenga (editor), "LDAP: Technical Specification Road Map", draft-ietf-ldapbis-roadmap-xx.txt (a work in progress).

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[X.680]   ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998 Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation

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

[LDAPIANA] K. Zeilenga, "IANA Considerations for LDAP", draft-ietf-ldapbis-xx.txt (a work in progress).

[ISO10646] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993.

[RFC2044] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996.

[Models] K. Zeilenga, "LDAP: The Models", draft-ietf-ldapbis-models-xx.txt (a work in progress).

[LDAPDN] K. Zeilenga (editor), "LDAP: String Representation of Distinguished Names", draft-ietf-ldapbis-dn-xx.txt, (a work in progress).

[Syntaxes] K. Dally (editor), "LDAP: Syntaxes", draft-ietf-ldapbis-syntaxes-xx.txt, (a work in progress).

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

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

[RFC2396] Berners-Lee, T., Fielding, R., and L. Masinter Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[AuthMeth] R. Harrison (editor), "LDAP: Authentication Methods", draft-ietf-ldapbis-authmeth-xx.txt, (a work in progress).

[RFC2222] Meyers, J., "Simple Authentication and Security Layer", RFC 2222, October 1997.

10. Editor's Address

Jim Sermersheim
Novell, Inc.
1800 South Novell Place
Provo, Utah 84606, USA
jimse@novell.com
+1 801 861-3088

Appendix A - LDAP Result Codes

This normative appendix details additional considerations regarding LDAP result codes and provides a brief, general description of each LDAP result code enumerated in Section 4.1.10.

Additional result codes MAY be defined for use with extensions. Client implementations SHALL treat any result code which they do not recognize as an unknown error condition.

A.1 Non-Error Result Codes

These result codes (called "non-error" result codes) do not indicate an error condition:

success(0),
compareTrue(6),

compareFalse(7),
referral(10), and
saslBindInProgress(14).

The success(0), compareTrue(6), and compare(7) result codes indicate successful completion (and, hence, are called to as "successful" result codes).

The referral(10) and saslBindInProgress(14) indicate the client is required to take additional action to complete the operation

A.2 Error Result Codes

A.3 Classes and Precedence of Error Result Codes

Result codes that indicate error conditions (and, hence, are called "error" result codes) fall into 6 classes. The following list specifies the precedence of error classes to be used when more than one error is detected [X511]:

1) Name Errors (codes 32 - 34, 36)
   - a problem related to a name (DN or RDN),
2) Update Errors (codes 64 - 69, 71)
   - a problem related to an update operation,
3) Attribute Errors (codes 16 - 21)
   - a problem related to a supplied attribute,
4) Security Errors (codes 8, 13, 48 - 50)
   - a security related problem,
5) Service Problem (codes 3, 4, 7, 11, 12, 51 - 54, 80)
   - a problem related to the provision of the service, and
6) Protocol Problem (codes 1, 2)
   - a problem related to protocol structure or semantics.

Server implementations SHALL NOT continue processing an operation after it has determined that an error is to be reported. If the server detects multiple errors simultaneously, the server SHOULD report the error with the highest precedence.

Existing LDAP result codes are described as follows:

success (0)

Indicates successful completion of an operation.

This result code is normally not returned by the compare operation, see compareFalse (5) and compareTrue (6).

operationsError (1)

Indicates that the operation is not properly sequenced with relation to other operations (of same or different type).

For example, this code is returned if the client attempts to Start TLS [RFC2830] while there are other operations outstanding or if TLS was already established.

For the bind operation only, the code indicates the server encountered an internal error.

Comment: Do we really want to carry this on

?

protocolError (2)

Indicates the server received data which has incorrect structure.

For bind operation only, the code may be resulted to indicate the server does not support the requested protocol version.

Comment: Do we really want to carry this on?

timeLimitExceeded (3)

Indicates that the time limit specified by the client was exceeded before the operation could be completed.

sizeLimitExceeded (4)

Indicates that the size limit specified by the client was exceeded before the operation could be completed.

compareFalse (5)

Indicates that the operation successfully completes and the assertion has evaluated to TRUE.

This result code is normally only returned by the compare operation.

compareTrue (6)

Indicates that the operation successfully completes and the assertion has evaluated to FALSE.

This result code is normally only returned by the compare operation.

authMethodNotSupported (7)

Indicates that authentication method or mechanism is not supported.

strongAuthRequired (8)

Except when returned in a Notice of Disconnect (see section 4.4.1), this indicates that the server requires the client to authentication using a strong(er) mechanism.

referral (10)

Indicates that a referral needs to be chased to complete the operation (see section 4.1.11).

adminLimitExceeded (11)

Indicates that an admnistrative limit has been exceeded.

unavailableCriticalExtension (12)

Indicates that server cannot perform a critical extension (see section 4.1.12).

confidentialityRequired (13)

Indicates that data confidentiality protections are required.

saslBindInProgress (14)

Indicates the server requires the client to send a new bind request, with the same sasl mechanism, to continue the authentication process (see section 4.2).

noSuchAttribute (16)

Indicates that the named entry does not contain the specified attribute or attribute value.

undefinedAttributeType (17)

Indicates that a request field contains an undefined attribute type.

inappropriateMatching (18)

Indicates that a request cannot be completed due to an inappropriate matching.

constraintViolation (19)

Indicates that the client supplied an attribute value which does not conform to constraints placed upon it by the data model.

For example, this code is returned when the multiple values are supplied to an attribute which has a SINGLE-VALUE constraint.

attributeOrValueExists (20)

Indicates that the client supplied an attribute or value to be added to an entry already exists.

invalidAttributeSyntax (21)

Indicates that a purported attribute value does not conform to the syntax of the attribute.

noSuchObject (32)

Indicates that the object does not exist in the DIT.

aliasProblem (33)

Indicates that an alias problem has occurred.

invalidDNSyntax (34)

Indicates that a LDAPDN or RelativeLDAPDN field (e.g. search base, target entry, ModifyDN newrdn, etc.) of a request does not conform to the required syntax or contains attribute values which do not conform to the syntax of the attribute's type.

aliasDereferencingProblem (36)

Indicates that a problem in dereferencing an alias.

inappropriateAuthentication (48)

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

invalidCredentials (49)

Indicates the supplied credentials are invalid.

insufficientAccessRights (50)

Indicates that the client does not have sufficient access rights to perform the operation.

busy (51)

Indicates that the server is busy.

unavailable (52)

Indicates that the server is shutting down or a subsystem necessary to complete the operation is offline.

unwillingToPerform (53)

Indicates that the server is unwilling to perform the operation.

loopDetect (54)

Indicates that the server has detected an internal loop.

namingViolation (64)

Indicates that the entry name violates naming restrictions.

objectClassViolation (65)

Indicates that the entry violates object class restrictions.

notAllowedOnNonLeaf (66)

Indicates that operation is inappropriately acting upon a non-leaf entry.

notAllowedOnRDN (67)

Indicates that the operation is inappropriately attempting to remove a value which forms the entry's relative distinguished name.

entryAlreadyExists (68)

Indicates that the request cannot be added fulfilled as the entry already exists.

objectClassModsProhibited (69)

Indicates that the attempt to modify the object class(es) of an entry objectClass attribute is prohibited.

For example, this code is returned when a when a client attempts to modify the structural object class of an entry.

affectsMultipleDSAs (71)

Indicates that the operation cannot be completed as it affects multiple servers (DSAs).

other (80)

Indicates the server has encountered an internal error.

Appendix B - Complete ASN.1 Definition

This appendix is normative.

```
 Lightweight-Directory-Access-Protocol-V3 DEFINITIONS
 IMPLICIT TAGS
 EXTENSIBILITY IMPLIED ::=
 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 -- UTF-8 encoded,
                             -- [ISO10646] characters

 LDAPOID ::= OCTET STRING -- Constrained to numericoid [Models]

 LDAPDN ::= LDAPString

 RelativeLDAPDN ::= LDAPString

 AttributeDescription ::= LDAPString
                          -- Constrained to attributedescription
                          -- [Models]

 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),
                     adminLimitExceeded           (11),
                     unavailableCriticalExtension (12),
                     confidentialityRequired      (13),
                     saslBindInProgress           (14),
                     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),
                                       -- 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 SIZE (1..MAX) OF Filter,
               or              [1] SET SIZE (1..MAX) 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,
             -- initial and final can occur at most once
               substrings      SEQUENCE OF CHOICE {
                       initial [0] AssertionValue,
                       any     [1] AssertionValue,
                       final   [2] AssertionValue } }

       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
```

Appendix C - Change History

<Note to RFC editor: This section is to be removed prior to RFC publication>

C.1 Changes made to RFC 2251:

C.1.1 Editorial

- Bibliography References: Changed all bibliography references to use a long name form for readability.
- Changed occurrences of "unsupportedCriticalExtension" "unavailableCriticalExtension"
- Fixed a small number of misspellings (mostly dropped letters).

C.1.2 Section 1

- Removed IESG note.

C.1.3 Section 9

- Added references to RFCs 1823, 2234, 2829 and 2830.

C.2 Changes made to draft-ietf-ldapbis-protocol-00.txt:

C.2.1 Section 4.1.6

- In the first paragraph, clarified what the contents of an AttributeValue are. There was confusion regarding whether or not

  an AttributeValue that is BER encoded (due to the "binary" option) is to be wrapped in an extra OCTET STRING.
- To the first paragraph, added wording that doesn't restrict other transfer encoding specifiers from being used. The previous wording only allowed for the string encoding and the ;binary encoding.
- To the first paragraph, added a statement restricting multiple options that specify transfer encoding from being present. This was never specified in the previous version and was seen as a potential interoperability problem.
- Added a third paragraph stating that the ;binary option is currently the only option defined that specifies the transfer encoding. This is for completeness.

C.2.2 Section 4.1.7

- Generalized the second paragraph to read "If an option specifying the transfer encoding is present in attributeDesc, the AssertionValue is encoded as specified by the option...". Previously, only the ;binary option was mentioned.

C.2.3 Sections 4.2, 4.9, 4.10

- Added alias dereferencing specifications. In the case of modDN, followed precedent set on other update operations (... alias is not dereferenced...) In the case of bind and compare stated that

servers SHOULD NOT dereference aliases. Specifications were added because they were missing from the previous version and caused interoperability problems. Concessions were made for bind and compare (neither should have ever allowed alias dereferencing) by using SHOULD NOT language, due to the behavior of some existing implementations.

C.2.4 Sections 4.5 and Appendix A

- Changed SubstringFilter.substrings.initial, any, and all from LDAPString to AssertionValue. This was causing an incompatibility with X.500 and confusion among other TS RFCs.

C.3 Changes made to draft-ietf-ldapbis-protocol-01.txt:

C.3.1 Section 3.4

- Reworded text surrounding subschemaSubentry to reflect that it is a single-valued attribute that holds the schema for the root DSE. Also noted that if the server masters entries that use differing schema, each entry's subschemaSubentry attribute must be interrogated. This may change as further fine-tuning is done to the data model.

C.3.2 Section 4.1.12

- Specified that the criticality field is only used for requests and not for unbind or abandon. Noted that it is ignored for all other operations.

C.3.3 Section 4.2

- Noted that Server behavior is undefined when the name is a null value, simple authentication is used, and a password is specified.

C.3.4 Section 4.2.(various)

- Changed "unauthenticated" to "anonymous" and "DN" and "LDAPDN" to "name"

C.3.5 Section 4.2.2

- Changed "there is no authentication or encryption being performed by a lower layer" to "the underlying transport service cannot guarantee confidentiality"

C.3.6 Section 4.5.2

- Removed all mention of ExtendedResponse due to lack of implementation.

C.4 Changes made to draft-ietf-ldapbis-protocol-02.txt:

C.4.1 Section 4

- Removed "typically" from "and is typically transferred" in the first paragraph. We know of no (and can conceive of no) case where this isn't true.
- Added "Section 5.1 specifies how the LDAP protocol is encoded." To the first paragraph. Added this cross reference for readability.
- Changed "version 3 " to "version 3 or later" in the second paragraph. This was added to clarify the original intent.
- Changed "protocol version" to "protocol versions" in the third paragraph. This attribute is multi-valued with the intent of holding all supported versions, not just one.

C.4.2 Section 4.1.8

- Changed "when transferred in protocol" to "when transferred from the server to the client" in the first paragraph. This is to clarify that this behavior only happens when attributes are being sent from the server.

C.4.3 Section 4.1.10

- Changed "servers will return responses containing fields of type LDAPResult" to "servers will return responses of LDAPResult or responses containing the components of LDAPResponse". This statement was incorrect and at odds with the ASN.1. The fix here reflects the original intent.
- Dropped '--new' from result codes ASN.1. This simplification in comments just reduces unneeded verbiage.

C.4.4 Section 4.1.11

- Changed "It contains a reference to another server (or set of servers)" to "It contains one or more references to one or more servers or services" in the first paragraph. This reflects the original intent and clarifies that the URL may point to non-LDAP services.

C.4.5 Section 4.1.12

- Changed "The server MUST be prepared" to "Implementations MUST be prepared" in the eighth paragraph to reflect that both client and server implementations must be able to handle this (as both parse controls).

C.4.6 Section 4.4

- Changed "One unsolicited notification is defined" to "One unsolicited notification (Notice of Disconnection) is defined" in the third paragraph. For clarity and readability.

C.4.7 Section 4.5.1

- Changed "checking for the existence of the objectClass attribute" to "checking for the presence of the objectClass attribute" in the last paragraph. This was done as a measure of consistency (we use the terms present and presence rather than exists and existence in search filters).

C.4.8 Section 4.5.3

- Changed "outstanding search operations to different servers," to "outstanding search operations" in the fifth paragraph as they may be to the same server. This is a point of clarification.

C.4.9 Section 4.6

- Changed "clients MUST NOT attempt to delete" to "clients MUST NOT attempt to add or delete" in the second to last paragraph.
- Change "using the "delete" form" to "using the "add" or "delete" form" in the second to last paragraph.

C.4.10 Section 4.7

- Changed "Clients MUST NOT supply the createTimestamp or creatorsName attributes, since these will be generated automatically by the server." to "Clients MUST NOT supply NO-USER-MODIFICATION attributes such as createTimestamp or creatorsName attributes, since these are provided by the server." in the definition of the attributes field. This tightens the language to reflect the original intent and to not leave a hole in which one could interpret the two attributes mentioned as the only non-writable attributes.

C.4.11 Section 4.11

- Changed "has been" to "will be" in the fourth paragraph. This clarifies that the server will (not has) abandon the operation.

C.5 Changes made to draft-ietf-ldapbis-protocol-03.txt:

C.5.1 Section 3.2.1

- Changed "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." to " An attribute is a description (a type and zero or more options) with one or more associated values. The attribute type governs whether the attribute can have multiple values, the syntax and matching rules used to construct and compare values of that attribute, and other functions. Options indicate modes of transfer and other

functions.". This points out that an attribute consists of both the type and options.

C.5.2 Section 4

- Changed "Section 5.1 specifies the encoding rules for the LDAP protocol" to "Section 5.1 specifies how the protocol is encoded and transferred."

C.5.3 Section 4.1.2

- Added ABNF for the textual representation of LDAPOID. Previously, there was no formal BNF for this construct.

C.5.4 Section 4.1.4

- Changed "This identifier may be written as decimal digits with components separated by periods, e.g. "2.5.4.10"" to "may be written as defined by ldapOID in section 4.1.2" in the second paragraph. This was done because we now have a formal BNF definition of an oid.

C.5.5 Section 4.1.5

- Changed the BNF for AttributeDescription to ABNF. This was done for readability and consistency (no functional changes involved).
- Changed "Options present in an AttributeDescription are never mutually exclusive." to "Options MAY be mutually exclusive. An AttributeDescription with mutually exclusive options is treated as an undefined attribute type." for clarity. It is generally understood that this is the original intent, but the wording could be easily misinterpreted.
- Changed "Any option could be associated with any AttributeType, although not all combinations may be supported by a server." to "Though any option or set of options could be associated with any AttributeType, the server support for certain combinations may be restricted by attribute type, syntaxes, or other factors.". This is to clarify the meaning of 'combination' (it applies both to combination of attribute type and options, and combination of options). It also gives examples of *why* they might be unsupported.

C.5.6 Section 4.1.11

- Changed the wording regarding 'equally capable' referrals to "If multiple URLs are present, the client assumes that any URL may be used to progress the operation.". The previous language implied that the server MUST enforce rules that it was practically incapable of. The new language highlights the original intent--that is, that any of the referrals may be used to progress the operation, there is no inherent 'weighting' mechanism.

C.5.7 Section 4.5.1 and Appendix A

- Added the comment "-- initial and final can occur at most once", to clarify this restriction.

C.5.8 Section 5.1

- Changed heading from "Mapping Onto BER-based Transport Services" to "Protocol Encoding".

C.5.9 Section 5.2.1

- Changed "The LDAPMessage PDUs" to "The encoded LDAPMessage PDUs" to point out that the PDUs are encoded before being streamed to TCP.

C.6 Changes made to draft-ietf-ldapbis-protocol-04.txt:

C.6.1 Section 4.5.1 and Appendix A

- Changed the ASN.1 for the and and or choices of Filter to have a lower range of 1. This was an omission in the original ASN.1

C.6.2 Various

- Fixed various typo's

C.7 Changes made to draft-ietf-ldapbis-protocol-05.txt:

C.7.1 Section 3.2.1

- Added "(as defined in Section 12.4.1 of [X.501])" to the fifth paragraph when talking about "operational attributes". This is because the term "operational attributes" is never defined. Alternately, we could drag a definition into the spec, for now, I'm just pointing to the reference in X.501.

C.7.2 Section 4.1.5

- Changed "And is also case insensitive" to "The entire AttributeDescription is case insensitive". This is to clarify whether we're talking about the entire attribute description, or just the options.

- Expounded on the definition of attribute description options. This doc now specifies a difference between transfer and tagging options and describes the semantics of each, and how and when subtyping rules apply. Now allow options to be transmitted in any order but disallow any ordering semantics to be implied. These changes are the result of ongoing input from an engineering team designed to deal with ambiguity issues surrounding attribute options.

C.7.3 Sections 4.1.5.1 and 4.1.6

- Refer to non "binary" transfer encodings as "native encoding" rather than "string" encoding to clarify and avoid confusion.

C.8 Changes made to draft-ietf-ldapbis-protocol-06.txt:

C.8.1 Title

- Changed to "LDAP: The Protocol" to be consisted with other working group documents

C.8.2 Abstract

- Moved above TOC to conform to new guidelines

- Reworded to make consistent with other WG documents.

- Moved 2119 conventions to "Conventions" section

C.8.3 Introduction

- Created to conform to new guidelines

C.8.4 Models

- Removed section. There is only one model in this document (Protocol Model)

C.8.5 Protocol Model

- Removed antiquated paragraph: "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."

- Removed antiquated paragraph concerning LDAP v1 and v2 and referrals.

C.8.6 Data Model

- Removed Section 3.2 and subsections. These have been moved to [Models]

C.8.7 Relationship to X.500

- Removed section. It has been moved to [Roadmap]

C.8.8 Server Specific Data Requirements

- Removed section. It has been moved to [Models]

C.8.9 Elements of Protocol

- Added "Section 5.1 specifies how the protocol is encoded and transferred." to the end of the first paragraph for reference.

- Reworded notes about extensibility, and now talk about implied extensibility and the use of ellipses in the ASN.1

- Removed references to LDAPv2 in third and fourth paragraphs.

C.8.10 Message ID

- Reworded second paragraph to "The message ID of a request MUST have a non-zero value different from the values of any other requests outstanding in the LDAP session of which this message is a part. The zero value is reserved for the unsolicited notification message." (Added notes about non-zero and the zero value).

C.8.11 String Types

- Removed ABNF for LDAPOID and added "Although an LDAPOID is encoded as an OCTET STRING, values are limited to the definition of numericoid given in Section 1.3 of [Models]."

C.8.12 Distinguished Name and Relative Distinguished Name

- Removed ABNF and referred to [Models] and [LDAPDN] where this is defined.

C.8.13 Attribute Type

- Removed sections. It's now in the [Models] doc.

C.8.14 Attribute Description

- Removed ABNF and aligned section with [Models]

- Moved AttributeDescriptionList here.

C.8.15 Transfer Options

- Added section and consumed much of old options language (while aligning with [Models]

C.8.16 Binary Transfer Option

- Clarified intent regarding exactly what is to be BER encoded.

- Clarified that clients must not expect ;binary when not asking for it (;binary, as opposed to ber encoded data).

C.8.17 Attribute

- Use the term "attribute description" in lieu of "type"

- Clarified the fact that clients cannot rely on any apparent ordering of attribute values.

C.8.18 LDAPResult

- To resultCode, added ellipses "..." to the enumeration to indicate extensibility. and added a note, pointing to [LDAPIANA]

- Removed error groupings ad refer to Appendix A.

C.8.19 Bind Operation

- Added "Prior to the BindRequest, the implied identity is anonymous. Refer to [AuthMeth] for the authentication-related semantics of this operation." to the first paragraph.

- Added ellipses "..." to AuthenticationChoice and added a note "This type is extensible as defined in Section 3.6 of [LDAPIANA]. Servers that do not support a choice supplied by a client will return authMethodNotSupported in the result code of the BindResponse."

- Simplified text regarding how the server handles unknown versions. Removed references to LDAPv2

C.8.20 Sequencing of the Bind Request

- Aligned with [AuthMeth] In particular, paragraphs 4 and 6 were removed, while a portion of 4 was retained (see C.8.9)

C.8.21 Authentication and other Security Service

- Section was removed. Now in [AuthMeth]

C.8.22 Continuation References in the Search Result

- Added "If the originating search scope was singleLevel, the scope part of the URL will be baseObject."

C.8.23 Security Considerations

- Removed reference to LDAPv2

C.8.24 Result Codes

- Added as normative appendix A

C.8.25 ASN.1

- Added EXTENSIBILITY IMPLIED

- Added a number of comments holding referenced to [Models] and [ISO10646].

- Removed AttributeType. It is not used.

C.9 Changes made to draft-ietf-ldapbis-protocol-07.txt:

- Removed all mention of transfer encodings and the binary attribute option

- Further alignment with [Models].

- Added extensibility ellipsis to protocol op choice

- In 4.1.1, clarified when connections may be dropped due to malformed PDUs

- Specified which matching rules and syntaxes are used for various filter items

C.10 Changes made to draft-ietf-ldapbis-protocol-07.txt:

C.10.1 Section 4.1.1.1:

- Clarified when it is and isn't appropriate to return an already used result code.

C.10.2 Section 4.1.11:

- Clarified that a control only applies to the message it's attached to.

- Explained that the criticality field is only applicable to certain request messages.

- Added language regarding the combination of controls.

C.10.3 Section 4.11:

- Explained that Abandon and Unbind cannot be abandoned, and illustrated how to determine whether an operation has been abandoned.

Appendix D - Outstanding Work Items

D.0 Integrate notational consistency agreements

- WG will discuss notation consistency. Once agreement happens, reconcile draft.

D.1 Integrate result codes draft.

- The result codes draft should be reconciled with this draft. Operation-specific instructions will reside with operations while the error-specific sections will be added as an appendix. Note

that there is a result codes appendix now. Still need to reconcile with each operation.

D.2 Verify references.

- Many referenced documents have changed. Ensure references and section numbers are correct.

D.3 Usage of Naming Context

- Make sure occurrences of "namingcontext" and "naming context" are consistent with [Models].

D.14 Section 4.1.12

- Specify whether or not servers are to advertise the OIDs of known response controls.

D.18 Section 4.2.3

- Change "operationsError" to "other" as a bind result code.

D.21 Section 4.5.1

- Make sure the use of "subordinates" in the derefInSearching definition is correct. See "derefInSearching" on list.

D.23 Section 4.5.3

- Add "Similarly, a server MUST NOT return a SearchResultReference when the scope of the search is baseObject. If a client receives such a SearchResultReference it MUST interpret is as a protocol error and MUST NOT follow it." to the first paragraph.

- Add "If the scope part of the LDAP URL is present, the client MUST use the new scope in its next request to progress the search. If the scope part is absent the client MUST use subtree scope to complete subtree searches and base scope to complete one level searches." to the third paragraph.

D.25 Section 4.6

- Resolve the meaning of "and is ignored if the attribute does not exist". See "modify: "non-existent attribute"" on the list.

D.27 Section 4.10

- Specify what happens when the attr is missing vs. attr isn't in schema. Also what happens if there's no equality matching rule.

D.30 Section 5.1

- Add "control and extended operation values" to last paragraph. See "LBER (BER Restrictions)" on list.

D.32 Section 6.1

- Add "that are used by those attributes" to the first paragraph.
- Add "Servers which support update operations MUST, and other servers SHOULD, support strong authentication mechanisms described in [RFC2829]." as a second paragraph.
- Add "Servers which provide access to sensitive information MUST, and other servers SHOULD support privacy protections such as those described in [RFC2829] and [RFC2830]." as a third paragraph.

D.33 Section 7

- Add "Servers which support update operations MUST, and other servers SHOULD, support strong authentication mechanisms described in [RFC2829]." as a fourth paragraph.
- Add "In order to automatically follow referrals, clients may need to hold authentication secrets. This poses significant privacy and security concerns and SHOULD be avoided." as a sixth paragraph.
- Add "This document provides a mechanism which clients may use to discover operational attributes. Those relying on security by obscurity should implement appropriate access controls to

  restricts access to operational attributes per local policy." as
  an eighth paragraph.
- Add "This document provides a mechanism which clients may use to
  discover operational attributes. Those relying on security by
  obscurity should implement appropriate access controls to
  restricts access to operational attributes per local policy." as
  an eighth paragraph.
- Add notes regarding DoS attack found by CERT advisories.

Full Copyright Statement