Network Working Group
Request for Comments: 4511
Obsoletes: 2251, 2830, 3771
Category: Standards Track

J. Sermersheim, Ed.
Novell, Inc.

June 2006

Lightweight Directory Access Protocol (LDAP): The Protocol

Status of This Memo

Copyright Notice

Abstract

This document describes the protocol elements, along with their semantics and encodings, of 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 3
   1.1. Relationship to Other LDAP Specifications 3
2. Conventions 3
3. Protocol Model 4
   3.1. Operation and LDAP Message Layer Relationship 5
4. Elements of Protocol 5
   4.1. Common Elements 5
        4.1.1. Message Envelope 6
        4.1.2. String Types 7
        4.1.3. Distinguished Name and Relative Distinguished Name ..8
        4.1.4. Attribute Descriptions 8
        4.1.5. Attribute Value 8
        4.1.6. Attribute Value Assertion 9
        4.1.7. Attribute and PartialAttribute 9
        4.1.8. Matching Rule Identifier 10
        4.1.9. Result Message 10
        4.1.10. Referral 12

4.1.11. Controls 14
4.2. Bind Operation 16
4.2.1. Processing of the Bind Request 17
4.2.2. Bind Response 18
4.3. Unbind Operation 18
4.4. Unsolicited Notification 19
4.4.1. Notice of Disconnection 19
4.5. Search Operation 20
4.5.1. Search Request 20
4.5.2. Search Result 27
4.5.3. Continuation References in the Search Result 28
4.6. Modify Operation 31
4.7. Add Operation 33
4.8. Delete Operation 34
4.9. Modify DN Operation 34
4.10. Compare Operation 36
4.11. Abandon Operation 36
4.12. Extended Operation 37
4.13. IntermediateResponse Message 39
4.13.1. Usage with LDAP ExtendedRequest and ExtendedResponse 40
4.13.2. Usage with LDAP Request Controls 40
4.14. StartTLS Operation 40
4.14.1. StartTLS Request 40
4.14.2. StartTLS Response 41
4.14.3. Removal of the TLS Layer 41
5. Protocol Encoding, Connection, and Transfer 42
5.1. Protocol Encoding 42
5.2. Transmission Control Protocol (TCP) 43
5.3. Termination of the LDAP session 43
6. Security Considerations 43
7. Acknowledgements 45
8. Normative References 46
9. Informative References 48
10. IANA Considerations 48
Appendix A. LDAP Result Codes 49
A.1. Non-Error Result Codes 49
A.2. Result Codes 49
Appendix B. Complete ASN.1 Definition 54
Appendix C. Changes 60
C.1. Changes Made to RFC 2251 60
C.2. Changes Made to RFC 2830 66
C.3. Changes Made to RFC 3771 66

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 the Lightweight Directory Access Protocol (LDAP), along with their semantics. Following the description of protocol elements, it describes the way in which the protocol elements are encoded and transferred.

1.1. Relationship to Other LDAP Specifications

This document is an integral part of the LDAP Technical Specification [RFC4510], which obsoletes the previously defined LDAP technical specification, RFC 3377, in its entirety.

This document, together with [RFC4510], [RFC4513], and [RFC4512], obsoletes RFC 2251 in its entirety. Section 3.3 is obsoleted by [RFC4510]. Sections 4.2.1 (portions) and 4.2.2 are obsoleted by [RFC4513]. Sections 3.2, 3.4, 4.1.3 (last paragraph), 4.1.4, 4.1.5, 4.1.5.1, 4.1.9 (last paragraph), 5.1, 6.1, and 6.2 (last paragraph) are obsoleted by [RFC4512]. The remainder of RFC 2251 is obsoleted by this document. Appendix C.1 summarizes substantive changes in the remainder.

This document obsoletes RFC 2830, Sections 2 and 4. The remainder of RFC 2830 is obsoleted by [RFC4513]. Appendix C.2 summarizes substantive changes to the remaining sections.

This document also obsoletes RFC 3771 in entirety.

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].

Character names in this document use the notation for code points and names from the Unicode Standard [Unicode]. For example, the letter "a" may be represented as either <U+0061> or <LATIN SMALL LETTER A>.

Note: a glossary of terms used in Unicode can be found in [Glossary]. Information on the Unicode character encoding model can be found in [CharModel].

The term "transport connection" refers to the underlying transport services used to carry the protocol exchange, as well as associations established by these services.

The term "TLS layer" refers to Transport Layer Security (TLS) services used in providing security services, as well as associations established by these services.

The term "SASL layer" refers to Simply Authentication and Security Layer (SASL) services used in providing security services, as well as associations established by these services.

The term "LDAP message layer" refers to the LDAP Message Protocol Data Unit (PDU) services used in providing directory services, as well as associations established by these services.

The term "LDAP session" refers to combined services (transport connection, TLS layer, SASL layer, LDAP message layer) and their associations.

See the table in Section 5 for an illustration of these four terms.

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 an operation, the server typically returns a response containing appropriate data to the requesting client.

Protocol operations are generally independent of one another. Each operation is processed as an atomic action, leaving the directory in a consistent state.

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 generally may be exchanged between a client and server in any order. If required, synchronous behavior may be controlled by client applications.

The core protocol operations defined in this document can be mapped to a subset of the X.500 (1993) Directory Abstract Service [X.511]. However, there is not a one-to-one mapping between LDAP operations and X.500 Directory Access Protocol (DAP) operations. Server implementations acting as a gateway to X.500 directories may need to make multiple DAP requests to service a single LDAP request.

3.1. Operation and LDAP Message Layer Relationship

Protocol operations are exchanged at the LDAP message layer. When the transport connection is closed, any uncompleted operations at the LDAP message layer are abandoned (when possible) or are completed without transmission of the response (when abandoning them is not possible). Also, when the transport connection is closed, the client MUST NOT assume that any uncompleted update operations have succeeded or failed.

4. Elements of Protocol

The protocol is described using Abstract Syntax Notation One ([ASN.1]) and is transferred using a subset of ASN.1 Basic Encoding Rules ([BER]). Section 5 specifies how the protocol elements are encoded and transferred.

In order to support future extensions to this protocol, extensibility is implied where it is allowed per ASN.1 (i.e., sequence, set, choice, and enumerated types are extensible). In addition, ellipses (...) have been supplied in ASN.1 types that are explicitly extensible as discussed in [RFC4520]. Because of the implied extensibility, clients and servers MUST (unless otherwise specified) ignore trailing SEQUENCE components whose tags they do not recognize.

Changes to the protocol other than through the extension mechanisms described here require a different version number. A client indicates the version it is using as part of the BindRequest, 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.

Clients may attempt to determine the protocol versions a server supports by reading the 'supportedLDAPVersion' attribute from the root DSE (DSA-Specific Entry) [RFC4512].

4.1. Common Elements

This section describes the LDAPMessage envelope Protocol Data Unit (PDU) 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,
          ...,
          intermediateResponse  IntermediateResponse },
     controls       [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 ..  maxInt)

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

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

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 messageID and the controls.

If the server receives an LDAPMessage 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 SHOULD return the Notice of Disconnection

described in Section 4.4.1, with the resultCode set to protocolError, and MUST immediately terminate the LDAP session as described in Section 5.3.

In other cases where the client or server cannot parse an LDAP PDU, it SHOULD abruptly terminate the LDAP session (Section 5.3) 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.

4.1.1.1. MessageID

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

The messageID of a request MUST have a non-zero value different from the messageID of any other request in progress in the same LDAP session. 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 messageID as an earlier request in the same LDAP session unless it can be determined that the server is no longer servicing the earlier request (e.g., after the final response is received, or a subsequent Bind completes). Otherwise, the behavior is undefined. For this purpose, note that Abandon and successfully abandoned operations do not send responses.

4.1.2. String Types

The LDAPString is a notational convenience to indicate that, although strings of LDAPString type encode as ASN.1 OCTET STRING types, the [ISO10646] character set (a superset of [Unicode]) is used, encoded following the UTF-8 [RFC3629] algorithm. Note that Unicode characters U+0000 through U+007F are the same as ASCII 0 through 127, respectively, and have the same single octet UTF-8 encoding. Other Unicode characters have a multiple octet UTF-8 encoding.

```
LDAPString ::= OCTET STRING -- UTF-8 encoded,
                            -- [ISO10646] 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 1.4 of [RFC4512].

```
LDAPOID ::= OCTET STRING -- Constrained to <numericoid>
                         -- [RFC4512]
```

For example,

```
1.3.6.1.4.1.1466.1.2.3
```

4.1.3. Distinguished Name and Relative Distinguished Name

An LDAPDN is defined to be the representation of a Distinguished Name (DN) after encoding according to the specification in [RFC4514].

```
LDAPDN ::= LDAPString
           -- Constrained to <distinguishedName> [RFC4514]
```

A RelativeLDAPDN is defined to be the representation of a Relative Distinguished Name (RDN) after encoding according to the specification in [RFC4514].

```
RelativeLDAPDN ::= LDAPString
                   -- Constrained to <name-component> [RFC4514]
```

4.1.4. Attribute Descriptions

The definition and encoding rules for attribute descriptions are defined in Section 2.5 of [RFC4512]. Briefly, an attribute description is an attribute type and zero or more options.

```
AttributeDescription ::= LDAPString
                         -- Constrained to <attributedescription>
                         -- [RFC4512]
```

4.1.5. Attribute Value

A field of type AttributeValue is an OCTET STRING containing an encoded attribute value. The attribute value is encoded according to the LDAP-specific encoding definition of its corresponding syntax. The LDAP-specific encoding definitions for different syntaxes and attribute types may be found in other documents and in particular [RFC4517].

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

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

Attribute values may be defined that have arbitrary and non-printable syntax. Implementations MUST NOT display or attempt to decode an attribute value if its syntax is not known. The implementation may attempt to discover the subschema of the source entry and to retrieve the descriptions of 'attributeTypes' from it [RFC4512].

Clients MUST only send attribute values in a request that are valid according to the syntax defined for the attributes.

4.1.6. Attribute Value Assertion

The AttributeValueAssertion (AVA) type definition is similar to the one in the X.500 Directory standards. It contains an attribute description and a matching rule ([RFC4512], Section 4.1.3) assertion value suitable for that type. Elements of this type are typically used to assert that the value in assertionValue matches a value of an attribute.

```
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 [RFC4517] for an example.

4.1.7. Attribute and PartialAttribute

Attributes and partial attributes consist of an attribute description and attribute values. A PartialAttribute allows zero values, while Attribute requires at least one value.

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

```
Attribute ::= PartialAttribute(WITH COMPONENTS {
     ...,
     vals (SIZE(1..MAX))})
```

No two of the attribute values may be equivalent as described by Section 2.2 of [RFC4512]. The set of attribute values is unordered. Implementations MUST NOT rely upon the ordering being repeatable.

4.1.8. Matching Rule Identifier

Matching rules are defined in Section 4.1.3 of [RFC4512]. A matching rule is identified in the protocol by the printable representation of either its <numericoid> or one of its short name descriptors [RFC4512], e.g., 'caseIgnoreMatch' or '2.5.13.2'.

```
MatchingRuleId ::= LDAPString
```

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 containing the elements found in LDAPResult to indicate the final status of the protocol operation request.

```
LDAPResult ::= SEQUENCE {
     resultCode         ENUMERATED {
          success                      (0),
          operationsError              (1),
          protocolError                (2),
          timeLimitExceeded            (3),
          sizeLimitExceeded            (4),
          compareFalse                 (5),
          compareTrue                  (6),
          authMethodNotSupported       (7),
          strongerAuthRequired         (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),
        ...  },
   matchedDN          LDAPDN,
   diagnosticMessage  LDAPString,
   referral           [3] Referral OPTIONAL }
```

The resultCode enumeration is extensible as defined in Section 3.8 of [RFC4520]. The meanings of the listed result codes are given in Appendix A. If a server detects multiple errors for an operation, only one result code is returned. The server should return the result code that best indicates the nature of the error encountered. Servers may return substituted result codes to prevent unauthorized disclosures.

The diagnosticMessage field of this construct may, at the server's option, be used to return a string containing a textual, human-readable diagnostic message (terminal control and page formatting characters should be avoided). As this diagnostic message is not standardized, implementations MUST NOT rely on the values returned. Diagnostic messages typically supplement the resultCode with additional information. If the server chooses not to return a textual diagnostic, the diagnosticMessage field MUST be empty.

For certain result codes (typically, but not restricted to noSuchObject, aliasProblem, invalidDNSyntax, and aliasDereferencingProblem), the matchedDN field is set (subject to access controls) to the name of the last entry (object or alias) used in finding the target (or base) object. This will be a truncated form of the provided name or, if an alias was dereferenced while attempting to locate the entry, of the resulting name. Otherwise, the matchedDN field is empty.

4.1.10. Referral

The referral result code indicates that the contacted server cannot or will not perform the operation and that one or more other servers may be able to. Reasons for this include:

- The target entry of the request is not held locally, but the server has knowledge of its possible existence elsewhere.

- The operation is restricted on this server -- perhaps due to a read-only copy of an entry to be modified.

The referral field is present in an LDAPResult if the resultCode is set to referral, and it is 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 URI 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 other servers would need to be contacted to complete the operation.

```
Referral ::= SEQUENCE SIZE (1..MAX) OF uri URI

URI ::= LDAPString     -- limited to characters permitted in
                       -- URIs
```

If the client wishes to progress the operation, it contacts one of the supported services found in the referral. If multiple URIs are present, the client assumes that any supported URI may be used to progress the operation.

Clients that follow 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 parameters. Some clients use a

counter that is incremented each time referral handling occurs for an operation, and these kinds of clients MUST be able to handle at least ten nested referrals while progressing the operation.

A URI for a server implementing LDAP and accessible via TCP/IP (v4 or v6) [RFC793][RFC791] is written as an LDAP URL according to [RFC4516].

Referral values that are LDAP URLs follow these rules:

- If an alias was dereferenced, the <dn> part of the LDAP URL MUST be present, with the new target object name.

- It is RECOMMENDED that the <dn> part be present to avoid ambiguity.

- If the <dn> part is present, the client uses this name in its next request to progress the operation, and if it is not present the client uses the same name as in the original request.

- Some servers (e.g., participating in distributed indexing) may provide a different filter in a URL of a referral for a Search operation.

- If the <filter> part of the LDAP URL is present, the client uses this filter in its next request to progress this Search, and if it is not present the client uses the same filter as it used for that Search.

- For Search, it is RECOMMENDED that the <scope> part be present to avoid ambiguity.

- If the <scope> part is missing, the scope of the original Search is used by the client to progress the operation.

- Other aspects of the new request may be the same as or different from the request that generated the referral.

Other kinds of URIs may be returned. The syntax and semantics of such URIs is left to future specifications. Clients may ignore URIs that they do not support.

UTF-8 encoded characters appearing in the string representation of a DN, search filter, or other fields of the referral value may not be legal for URIs (e.g., spaces) and MUST be escaped using the % method in [RFC3986].

4.1.11. Controls

Controls provide a mechanism whereby the semantics and arguments of existing LDAP operations may be extended. One or more controls may be attached to a single LDAP message. A control only affects the semantics of the message it is attached to.

Controls sent by clients are termed 'request controls', and those sent by servers are termed 'response controls'.

```
Controls ::= SEQUENCE OF control Control

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

The controlType field is the dotted-decimal representation of an OBJECT IDENTIFIER that uniquely identifies the control. This provides unambiguous naming of controls. Often, response control(s) solicited by a request control share controlType values with the request control.

The criticality field only has meaning in controls attached to request messages (except UnbindRequest). For controls attached to response messages and the UnbindRequest, the criticality field SHOULD be FALSE, and MUST be ignored by the receiving protocol peer. A value of TRUE indicates that it is unacceptable to perform the operation without applying the semantics of the control. Specifically, the criticality field is applied as follows:

- If the server does not recognize the control type, determines that it is not appropriate for the operation, or is otherwise unwilling to perform the operation with the control, and if the criticality field is TRUE, the server MUST NOT perform the operation, and for operations that have a response message, it MUST return with the resultCode set to unavailableCriticalExtension.

- If the server does not recognize the control type, determines that it is not appropriate for the operation, or is otherwise unwilling to perform the operation with the control, and if the criticality field is FALSE, the server MUST ignore the control.

- Regardless of criticality, if a control is applied to an operation, it is applied consistently and impartially to the entire operation.

The controlValue may contain information associated with the controlType. Its format is defined by the specification of 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 that is associated with a control of its type. When a controlValue is defined in terms of ASN.1, and BER-encoded according to Section 5.1, it also follows the extensibility rules in Section 4.

Servers list the controlType of request controls they recognize in the 'supportedControl' attribute in the root DSE (Section 5.1 of [RFC4512]).

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. When a combination of controls is encountered whose semantics are invalid, not specified (or not known), the message is considered not well-formed; thus, the operation fails with protocolError. Controls with a criticality of FALSE may be ignored in order to arrive at a valid combination. Additionally, unless order-dependent semantics are given in a specification, the order of a combination of controls in the SEQUENCE is ignored. Where the order is to be ignored but cannot be ignored by the server, the message is considered not well-formed, and the operation fails with protocolError. Again, controls with a criticality of FALSE may be ignored in order to arrive at a valid combination.

This document does not specify any controls. Controls may be specified in other documents. Documents detailing control extensions are to provide for each control:

- the OBJECT IDENTIFIER assigned to the control,

- direction as to what value the sender should provide for the criticality field (note: the semantics of the criticality field are defined above should not be altered by the control's specification),

- whether the controlValue field is present, and if so, the format of its contents,

- the semantics of the control, and

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

4.2. Bind Operation

The function of the Bind operation is to allow authentication information to be exchanged between the client and server. The Bind operation should be thought of as the "authenticate" operation. Operational, authentication, and security-related semantics of this operation are given in [RFC4513].

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 }
```

Fields of the BindRequest are:

- version: A version number indicating the version of the protocol to be used at the LDAP message layer. This document describes version 3 of the protocol. There is no version negotiation. The client sets this field to the version it desires. If the server does not support the specified version, it MUST respond with a BindResponse where the resultCode is set to protocolError.

- name: If not empty, 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 ([RFC4513], Section 5.1) or when using SASL [RFC4422] authentication ([RFC4513], Section 5.2). Where the server attempts to locate the named object, it SHALL NOT perform alias dereferencing.

- authentication: Information used in authentication. This type is extensible as defined in Section 3.7 of [RFC4520]. Servers that do not support a choice supplied by a client return a BindResponse with the resultCode set to authMethodNotSupported.

Textual passwords (consisting of a character sequence with a known character set and encoding) transferred to the server using the simple AuthenticationChoice SHALL be transferred as UTF-8 [RFC3629] encoded [Unicode]. Prior to transfer, clients SHOULD prepare text passwords as "query" strings by applying the SASLprep [RFC4013] profile of the stringprep [RFC3454] algorithm. Passwords consisting of other data (such as random octets) MUST NOT be altered. The determination of whether a password is textual is a local client matter.

4.2.1. Processing of the Bind Request

Before processing a BindRequest, all uncompleted operations MUST either complete or be abandoned. The server may either wait for the uncompleted operations to complete, or abandon them. The server then proceeds to authenticate the client in either a single-step or multi-step Bind process. Each step requires the server to return a BindResponse to indicate the status of authentication.

After sending a BindRequest, clients MUST NOT send further LDAP PDUs until receiving the BindResponse. Similarly, servers SHOULD NOT process or respond to requests received while processing a BindRequest.

If the client did not bind before sending a request and receives an operationsError to that request, it may then send a BindRequest. If this also fails or the client chooses not to bind on the existing LDAP session, it may terminate the LDAP session, re-establish it, and begin again by first sending a BindRequest. This will aid in interoperating with servers implementing other versions of LDAP.

Clients may send multiple Bind requests to change the authentication and/or security associations or to complete a multi-stage Bind process. Authentication from earlier binds is subsequently ignored.

For some SASL authentication mechanisms, it may be necessary for the client to invoke the BindRequest multiple times ([RFC4513], Section 5.2). 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 the resultCode set to authMethodNotSupported. This will allow the client to abort a negotiation if it wishes to try again with the same SASL mechanism.

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.

A successful Bind operation is indicated by a BindResponse with a resultCode set to success. Otherwise, an appropriate result code is set in the BindResponse. For BindResponse, the protocolError result code may be used to indicate that the version number supplied by the client is unsupported.

If the client receives a BindResponse where the resultCode is set to protocolError, it is to assume that the server does not support this version of LDAP. While the client may be able proceed with another version of this protocol (which may or may not require closing and re-establishing the transport connection), how to proceed with another version of this protocol is beyond the scope of this document. Clients that are unable or unwilling to proceed SHOULD terminate the LDAP session.

The serverSaslCreds field is 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 SHALL NOT be included in the BindResponse.

4.3. Unbind Operation

The function of the Unbind operation is to terminate an LDAP session. The Unbind operation is not the antithesis of the Bind operation as the name implies. The naming of these operations are historical. The Unbind operation should be thought of as the "quit" operation.

The Unbind operation is defined as follows:

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

The client, upon transmission of the UnbindRequest, and the server, upon receipt of the UnbindRequest, are to gracefully terminate the LDAP session as described in Section 5.3. Uncompleted operations are handled as specified in Section 3.1.

4.4. Unsolicited Notification

An unsolicited notification is an LDAPMessage sent from the server to the client that 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 LDAP session 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 zero and protocolOp is set to the extendedResp choice using the ExtendedResponse type (See Section 4.12). The responseName field of the ExtendedResponse always contains an LDAPOID that is unique for this notification.

One unsolicited notification (Notice of Disconnection) is defined in this document. The specification of an unsolicited notification consists of:

- the OBJECT IDENTIFIER assigned to the notification (to be specified in the responseName,

- the format of the contents of the responseValue (if any),

- the circumstances which will cause the notification to be sent, and

- the semantics of the message.

4.4.1. Notice of Disconnection

This notification may be used by the server to advise the client that the server is about to terminate the LDAP session on its own initiative. This notification is intended to assist clients in distinguishing between an exceptional server condition and a transient network failure. Note that this notification is not a response to an Unbind requested by the client. Uncompleted operations are handled as specified in Section 3.1.

The responseName is 1.3.6.1.4.1.1466.20036, the responseValue field is absent, and the resultCode is used to indicate the reason for the disconnection. When the strongerAuthRequired resultCode is returned with this message, it indicates that the server has detected that an established security association between the client and server has unexpectedly failed or been compromised.

Upon transmission of the Notice of Disconnection, the server gracefully terminates the LDAP session as described in Section 5.3.

4.5. Search Operation

The Search operation is used to request a server to return, subject to access controls and other restrictions, a set of entries matching a complex search criterion. This can be used to read attributes from a single entry, from entries immediately subordinate to a particular entry, or from 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      AttributeSelection }

AttributeSelection ::= SEQUENCE OF selector LDAPString
                -- The LDAPString is constrained to
                -- <attributeSelector> in Section 4.5.1.8

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

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

Note that an X.500 "list"-like operation can be emulated by the client requesting a singleLevel 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 baseObject Search operation with the same filter. A server that 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.

4.5.1.1. SearchRequest.baseObject

The name of the base object entry (or possibly the root) relative to which the Search is to be performed.

4.5.1.2. SearchRequest.scope

Specifies the scope of the Search to be performed. The semantics (as described in [X.511]) of the defined values of this field are:

baseObject: The scope is constrained to the entry named by baseObject.

singleLevel: The scope is constrained to the immediate subordinates of the entry named by baseObject.

wholeSubtree: The scope is constrained to the entry named by baseObject and to all its subordinates.

4.5.1.3. SearchRequest.derefAliases

An indicator as to whether or not alias entries (as defined in [RFC4512]) are to be dereferenced during stages of the Search operation.

The act of dereferencing an alias includes recursively dereferencing aliases that refer to aliases.

Servers MUST detect looping while dereferencing aliases in order to prevent denial-of-service attacks of this nature.

The semantics of the defined values of this field are:

neverDerefAliases: Do not dereference aliases in searching or in locating the base object of the Search.

derefInSearching: While searching subordinates of the base object, dereference any alias within the search scope. Dereferenced objects become the vertices of further search scopes where the Search operation is also applied. If the search scope is wholeSubtree, the Search continues in the subtree(s) of any dereferenced object. If the search scope is singleLevel, the search is applied to any dereferenced objects and is not applied to their subordinates. Servers SHOULD eliminate duplicate entries that arise due to alias dereferencing while searching.

derefFindingBaseObj: Dereference aliases in locating the base object of the Search, but not when searching subordinates of the base object.

derefAlways: Dereference aliases both in searching and in locating the base object of the Search.

4.5.1.4. SearchRequest.sizeLimit

A size limit that restricts the maximum number of entries to be returned as a result of the Search. A value of zero in this field indicates that no client-requested size limit restrictions are in effect for the Search. Servers may also enforce a maximum number of entries to return.

4.5.1.5. SearchRequest.timeLimit

A time limit that restricts the maximum time (in seconds) allowed for a Search. A value of zero in this field indicates that no client-requested time limit restrictions are in effect for the Search. Servers may also enforce a maximum time limit for the Search.

4.5.1.6. SearchRequest.typesOnly

An indicator as to whether Search results are to contain both attribute descriptions and values, or just attribute descriptions. Setting this field to TRUE causes only attribute descriptions (and not values) to be returned. Setting this field to FALSE causes both attribute descriptions and values to be returned.

4.5.1.7. SearchRequest.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 were explicit.)

A server MUST evaluate filters according to the three-valued logic of [X.511] (1993), Clause 7.8.1. In summary, a filter is evaluated to "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 Undefined otherwise. A filter of the "or" choice is FALSE if all 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.

A filter item evaluates to Undefined when the server would not be able to determine whether the assertion value matches an entry. Examples include:

- An attribute description in an equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch, or extensibleMatch filter is not recognized by the server.

- The attribute type does not define the appropriate matching rule.

- A MatchingRuleId in the extensibleMatch is not recognized by the server or is not valid for the attribute type.

- The type of filtering requested is not implemented.

- The assertion value is invalid.

For example, if a server did not recognize the attribute type shoeSize, the filters (shoeSize=*), (shoeSize=12), (shoeSize>=12), and (shoeSize<=12) would each evaluate to Undefined.

Servers MUST NOT return errors if attribute descriptions or matching rule ids are not recognized, assertion values are invalid, or the assertion syntax is not supported. More details of filter processing are given in Clause 7.8 of [X.511].

4.5.1.7.1. SearchRequest.filter.equalityMatch

The matching rule for an equalityMatch filter is defined by the EQUALITY matching rule for the attribute type or subtype. The filter is TRUE when the EQUALITY rule returns TRUE as applied to the attribute or subtype and the asserted value.

4.5.1.7.2. SearchRequest.filter.substrings

There SHALL be at most one 'initial' and at most one 'final' in the 'substrings' of a SubstringFilter. If 'initial' is present, it SHALL be the first element of 'substrings'. If 'final' is present, it SHALL be the last element of 'substrings'.

The matching rule for an AssertionValue in a substrings filter item is defined by the SUBSTR matching rule for the attribute type or subtype. The filter is TRUE when the SUBSTR rule returns TRUE as applied to the attribute or subtype and the asserted value.

Note that the AssertionValue in a substrings filter item conforms to the assertion syntax of the EQUALITY matching rule for the attribute type rather than to the assertion syntax of the SUBSTR matching rule for the attribute type. Conceptually, the entire SubstringFilter is converted into an assertion value of the substrings matching rule prior to applying the rule.

4.5.1.7.3. SearchRequest.filter.greaterOrEqual

The matching rule for a greaterOrEqual filter is defined by the ORDERING matching rule for the attribute type or subtype. The filter is TRUE when the ORDERING rule returns FALSE as applied to the attribute or subtype and the asserted value.

4.5.1.7.4. SearchRequest.filter.lessOrEqual

The matching rules for a lessOrEqual filter are defined by the ORDERING and EQUALITY matching rules for the attribute type or subtype. The filter is TRUE when either the ORDERING or EQUALITY rule returns TRUE as applied to the attribute or subtype and the asserted value.

4.5.1.7.5. SearchRequest.filter.present

A present filter is TRUE when there is an attribute or subtype of the specified attribute description present in an entry, FALSE when no attribute or subtype of the specified attribute description is present in an entry, and Undefined otherwise.

4.5.1.7.6. SearchRequest.filter.approxMatch

An approxMatch filter is TRUE when there is a value of the attribute type or subtype for which some locally-defined approximate matching algorithm (e.g., spelling variations, phonetic match, etc.) returns TRUE. If a value matches for equality, it also satisfies an approximate match. If approximate matching is not supported for the attribute, this filter item should be treated as an equalityMatch.

4.5.1.7.7. SearchRequest.filter.extensibleMatch

The fields of the extensibleMatch filter item are evaluated as follows:

- If the matchingRule field is absent, the type field MUST be present, and an equality match is performed for that type.

- If the type field is absent and the matchingRule is present, the matchValue is compared against all attributes in an entry that support that matchingRule.

- If the type field is present and the matchingRule is present, the matchValue is compared against the specified attribute type and its subtypes.

- If the dnAttributes field is set to TRUE, the match is additionally applied against all the AttributeValueAssertions in an entry's distinguished name, and it evaluates to TRUE if there is at least one attribute or subtype in the distinguished name for which the filter item evaluates to TRUE. The dnAttributes field is present to alleviate the need for multiple versions of generic matching rules (such as word matching), where one applies to entries and another applies to entries and DN attributes as well.

The matchingRule used for evaluation determines the syntax for the assertion value. Once the matchingRule and attribute(s) have been determined, the filter item evaluates to TRUE if it matches at least one attribute type or subtype in the entry, FALSE if it does not match any attribute type or subtype in the entry, and Undefined if the matchingRule is not recognized, the matchingRule is unsuitable for use with the specified type, or the assertionValue is invalid.

4.5.1.8. SearchRequest.attributes

A selection list of the attributes to be returned from each entry that matches the search filter. Attributes that are subtypes of listed attributes are implicitly included. LDAPString values of this field are constrained to the following Augmented Backus-Naur Form (ABNF) [RFC4234]:

```
attributeSelector = attributedescription / selectorspecial

selectorspecial = noattrs / alluserattrs

noattrs = %x31.2E.31 ; "1.1"

alluserattrs = %x2A ; asterisk ("*")
```

The <attributedescription> production is defined in Section 2.5 of [RFC4512].

There are three special cases that may appear in the attributes selection list:

1. An empty list with no attributes requests the return of all user attributes.

2. A list containing "*" (with zero or more attribute descriptions) requests the return of all user attributes in addition to other listed (operational) attributes.

3. A list containing only the OID "1.1" indicates that no attributes are to be returned. If "1.1" is provided with other attributeSelector values, the "1.1" attributeSelector is ignored. This OID was chosen because it does not (and can not) correspond to any attribute in use.

Client implementors should note that even if all user attributes are requested, some attributes and/or attribute values 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. Operational attributes are described in [RFC4512].

Attributes are returned at most once in an entry. If an attribute description is named more than once in the list, the subsequent names are ignored. If an attribute description in the list is not recognized, it is ignored by the server.

4.5.2. Search Result

The results of the Search operation are returned as zero or more SearchResultEntry and/or SearchResultReference messages, followed by a single SearchResultDone message.

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

PartialAttributeList ::= SEQUENCE OF
                         partialAttribute PartialAttribute

SearchResultReference ::= [APPLICATION 19] SEQUENCE
                              SIZE (1..MAX) OF uri URI

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

Each SearchResultEntry represents an entry found during the Search. Each SearchResultReference represents an area not yet explored during the Search. The SearchResultEntry and SearchResultReference messages may come in any order. Following all the SearchResultReference and SearchResultEntry responses, the server returns a SearchResultDone response, which contains an indication of success or details any errors that have occurred.

Each entry returned in a SearchResultEntry will contain all appropriate attributes as specified in the attributes field of the Search Request, subject to access control and other administrative policy. Note that the PartialAttributeList may hold zero elements.

This may happen when none of the attributes of an entry were requested or could be returned. Note also that the partialAttribute vals set may hold zero elements. This may happen when typesOnly is requested, access controls prevent the return of values, or other reasons.

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 this is permitted by access control.

If the server's schema defines short names [RFC4512] for an attribute type, then the server SHOULD use one of those names in attribute descriptions for that attribute type (in preference to using the <numericoid> [RFC4512] format of the attribute type's object identifier). The server SHOULD NOT use the short name if that name is known by the server to be ambiguous, or if it is otherwise likely to cause interoperability problems.

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 or unwilling to search one or more non-local entries, the server may return one or more SearchResultReference messages, each containing a reference to another set of servers for continuing the operation. A server MUST NOT return any SearchResultReference messages if it has not located the baseObject and thus has not searched any entries. In this case, it would return a SearchResultDone containing either a referral or noSuchObject result code (depending on the server's knowledge of the entry named in the baseObject).

If a server holds a copy or partial copy of the subordinate naming context (Section 5 of [RFC4512]), it may use the search filter to determine whether or not to return a SearchResultReference response. Otherwise, SearchResultReference responses are always returned when in scope.

The SearchResultReference is of the same data type as the Referral.

If the client wishes to progress the Search, it issues a new Search operation for each SearchResultReference that is returned. If multiple URIs are present, the client assumes that any supported URI may be used to progress the operation.

Clients that follow search continuation references MUST ensure that they do not loop between servers. They MUST NOT repeatedly contact the same server for the same request with the same parameters. Some clients use a counter that is incremented each time search result reference handling occurs for an operation, and these kinds of clients MUST be able to handle at least ten nested referrals while progressing the operation.

Note that the Abandon operation described in Section 4.11 applies only to a particular operation sent at the LDAP message layer between a client and server. The client must individually abandon subsequent Search operations it wishes to.

A URI for a server implementing LDAP and accessible via TCP/IP (v4 or v6) [RFC793][RFC791] is written as an LDAP URL according to [RFC4516].

SearchResultReference values that are LDAP URLs follow these rules:

- The <dn> part of the LDAP URL MUST be present, with the new target object name. The client uses this name when following the reference.

- Some servers (e.g., participating in distributed indexing) may provide a different filter in the LDAP URL.

- If the <filter> part of the LDAP URL is present, the client uses this filter in its next request to progress this Search, and if it is not present the client uses the same filter as it used for that Search.

- If the originating search scope was singleLevel, the <scope> part of the LDAP URL will be "base".

- It is RECOMMENDED that the <scope> part be present to avoid ambiguity. In the absence of a <scope> part, the scope of the original Search request is assumed.

- Other aspects of the new Search request may be the same as or different from the Search request that generated the SearchResultReference.

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

Other kinds of URIs may be returned. The syntax and semantics of such URIs is left to future specifications. Clients may ignore URIs that they do not support.

UTF-8-encoded characters appearing in the string representation of a DN, search filter, or other fields of the referral value may not be legal for URIs (e.g., spaces) and MUST be escaped using the % method in [RFC3986].

4.5.3.1. Examples

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 both LDAP servers (hostb) and (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 wholeSubtree 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??sub
  ldap://hostc/OU=People,DC=Example,DC=NET??sub }
SearchResultReference {
  ldap://hostd/OU=Roles,DC=Example,DC=NET??sub }
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 request 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??sub }
SearchResultReference {
  ldap://hostf/OU=Consultants,OU=People,DC=Example,DC=NET??sub }
SearchResultDone (success)
```

Similarly, if a singleLevel Search of <DC=Example,DC=NET> is requested to the contacted server, it may return the following:

```
SearchResultEntry for CN=Manager,DC=Example,DC=NET
SearchResultReference {
  ldap://hostb/OU=People,DC=Example,DC=NET??base
  ldap://hostc/OU=People,DC=Example,DC=NET??base }
SearchResultReference {
  ldap://hostd/OU=Roles,DC=Example,DC=NET??base }
SearchResultDone (success)
```

If the contacted server does not hold the base object for the Search, but has knowledge of its possible location, then it may return a referral to the client. In this case, if the client requests a subtree Search of <DC=Example,DC=ORG> to hosta, the server returns a SearchResultDone containing a referral.

```
SearchResultDone (referral) {
  ldap://hostg/DC=Example,DC=ORG??sub }
```

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,
     changes         SEQUENCE OF change SEQUENCE {
          operation       ENUMERATED {
               add     (0),
               delete  (1),
               replace (2),
               ...  },
          modification    PartialAttribute } }
```

Fields of the Modify Request are:

- object: The value of this field contains the name of the entry to be modified. The server SHALL NOT perform any alias dereferencing in determining the object to be modified.

- changes: A list of modifications to be performed on the entry. The entire list of modifications MUST be performed in the order they are listed as a single atomic operation. While individual modifications may violate certain aspects of the directory schema (such as the object class definition and Directory Information Tree (DIT) content rule), the resulting entry after the entire list of modifications is performed MUST conform to the requirements of the directory model and controlling schema [RFC4512].

  - operation: Used to specify the type of modification being performed. Each operation type acts on the following modification. The values of this field have the following semantics, respectively:

    add: add values listed to the modification attribute, creating the attribute if necessary.

delete: delete values listed from the modification attribute. If no values are listed, or if all current values of the attribute are listed, the entire attribute is removed.

replace: replace all existing values of the modification 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 it is ignored if the attribute does not exist.

- modification: A PartialAttribute (which may have an empty SET of vals) used to hold the attribute type or attribute type and values being modified.

Upon receipt of a Modify Request, the server attempts to perform the necessary modifications to the DIT and returns the result in a Modify Response, defined as follows:

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

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. 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. Whether or not the modification was applied cannot be determined by the client if the Modify Response was not received (e.g., the LDAP session was terminated or the Modify operation was abandoned).

Servers MUST ensure that entries conform to user and system schema rules or other data model constraints. The Modify operation cannot be used to remove from an entry any of its distinguished values, i.e., those values which form the entry's relative distinguished name. An attempt to do so will result in the server returning the notAllowedOnRDN result code. The Modify DN operation described in Section 4.9 is used to rename an entry.

For attribute types that specify no equality matching, the rules in Section 2.5.1 of [RFC4512] are followed.

Note that due to the simplifications made in LDAP, there is not a direct mapping of the changes in an LDAP ModifyRequest onto the changes 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 attribute Attribute
```

Fields of the Add Request are:

- entry: the name of the entry to be added. The server SHALL NOT dereference any aliases in locating the entry to be added.

- attributes: the list of attributes that, along with those from the RDN, make up the content of the entry being added. Clients MAY or MAY NOT include the RDN attribute(s) in this list. Clients MUST NOT supply NO-USER-MODIFICATION attributes such as the createTimestamp or creatorsName attributes, since the server maintains these automatically.

Servers MUST ensure that entries conform to user and system schema rules or other data model constraints. For attribute types that specify no equality matching, the rules in Section 2.5.1 of [RFC4512] are followed (this applies to the naming attribute in addition to any multi-valued attributes being added).

The entry named in the entry field of the AddRequest MUST NOT exist for the AddRequest to succeed. The immediate superior (parent) of an object or alias 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 noSuchObject result code with the matchedDN field containing <DC=NET>.

Upon receipt of an Add Request, a server will attempt to add the requested entry. 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 has been added to 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 name of the entry to be deleted. The server SHALL NOT dereference aliases while resolving the name of the target entry to be removed.

Only leaf entries (those with no subordinate entries) can be deleted with this operation.

Upon receipt of a Delete Request, a server will attempt to perform the entry removal requested and return the result in the Delete Response defined as follows:

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

4.9. Modify DN Operation

The Modify DN operation allows a client to change the Relative Distinguished Name (RDN) of an entry in the Directory and/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 }
```

Fields of the Modify DN Request are:

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

- newrdn: the new RDN of the entry. The value of the old RDN is supplied when moving the entry to a new superior without changing its RDN. Attribute values of the new RDN not matching any attribute value of the entry are added to the entry, and an appropriate error is returned if this fails.

- deleteoldrdn: a boolean field 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 name of an existing object entry that becomes the immediate superior (parent) of the existing entry.

The server SHALL NOT dereference any aliases in locating the objects named in entry or newSuperior.

Upon receipt of a ModifyDNRequest, a server will attempt to perform the name change and return the result in the Modify DN Response, defined as follows:

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

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

Servers MUST ensure that entries conform to user and system schema rules or other data model constraints. For attribute types that specify no equality matching, the rules in Section 2.5.1 of [RFC4512] are followed (this pertains to newrdn and deleteoldrdn).

The object named in newSuperior 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 noSuchObject result code with the matchedDN field containing <DC=NET>.

If the deleteoldrdn field is TRUE, the attribute values forming the old RDN (but not the new RDN) are deleted from the entry. If the deleteoldrdn field is FALSE, the attribute values forming the old RDN will be retained as non-distinguished attribute values of the entry.

Note that X.500 restricts the ModifyDN operation to affect only entries that are contained within a single server. If the LDAP server is mapped onto DAP, then this restriction will apply, and the affectsMultipleDSAs result code 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 or between naming contexts.

4.10. Compare Operation

The Compare operation allows a client to compare an assertion value with the values of a particular attribute in a particular entry in the Directory. The Compare Request is defined as follows:

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

Fields of the Compare Request are:

- entry: the name of the entry to be compared. The server SHALL NOT dereference any aliases in locating the entry to be compared.

- ava: holds the attribute value assertion to be compared.

Upon receipt of a Compare Request, a server will attempt to perform the requested comparison and return the result in the Compare Response, defined as follows:

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

The resultCode is set to compareTrue, compareFalse, or an appropriate error. compareTrue indicates that the assertion value in the ava field matches a value of the attribute or subtype according to the attribute's EQUALITY matching rule. compareFalse indicates that the assertion value in the ava field and the values of the attribute or subtype did not match. Other result codes indicate either that the result of the comparison was Undefined (Section 4.5.1.7), or that some error occurred.

Note that some directory systems may establish access controls that permit the values of certain attributes (such as userPassword) to be compared but not interrogated by other means.

4.11. Abandon Operation

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

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

The MessageID is that of an operation that was requested earlier at this LDAP message layer. The Abandon request itself has its own MessageID. This is distinct from the MessageID of the earlier operation being abandoned.

There is no response defined in the Abandon operation. Upon receipt of an AbandonRequest, the server MAY abandon the operation identified by the MessageID. Since the client cannot tell the difference between a successfully abandoned operation and an uncompleted operation, the application of the Abandon operation is limited to uses where the client does not require an indication of its outcome.

Abandon, Bind, Unbind, and StartTLS operations cannot be abandoned.

In the event that a server receives an Abandon Request on a Search operation in the midst of transmitting responses to the Search, that server MUST cease transmitting entry responses to the abandoned request immediately, and it MUST NOT send the SearchResultDone. Of course, the server MUST ensure that only properly encoded LDAPMessage PDUs are transmitted.

The ability to abandon other (particularly update) operations is at the discretion of the server.

Clients should not send Abandon requests for the same operation multiple times, and they MUST also be prepared to receive results from operations they have abandoned (since these might have been in transit when the Abandon was requested or might not be able to be abandoned).

Servers MUST discard Abandon requests for messageIDs they do not recognize, for operations that cannot be abandoned, and for operations that have already been abandoned.

4.12. Extended Operation

The Extended operation allows additional operations to be defined for services not already available in the protocol; for example, to Add operations to install transport layer security (see Section 4.14).

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 Extended operation consists of an Extended request and an Extended response.

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

The requestName is a dotted-decimal representation of the unique 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 an ExtendedResponse.

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

The responseName field, when present, contains an LDAPOID that is unique for this extended operation or response. This field is optional (even when the extension specification defines an LDAPOID for use in this field). The field will be absent whenever the server is unable or unwilling to determine the appropriate LDAPOID to return, for instance, when the requestName cannot be parsed or its value is not recognized.

Where the requestName is not recognized, the server returns protocolError. (The server may return protocolError in other cases.)

The requestValue and responseValue fields contain information associated with the operation. The format of these fields is defined by the specification of the Extended operation. Implementations MUST be prepared to handle arbitrary contents of these fields, including zero bytes. Values that are defined in terms of ASN.1 and BER-encoded according to Section 5.1 also follow the extensibility rules in Section 4.

Servers list the requestName of Extended Requests they recognize in the 'supportedExtension' attribute in the root DSE (Section 5.1 of [RFC4512]).

Extended operations may be specified in other documents. The specification of an Extended operation consists of:

- the OBJECT IDENTIFIER assigned to the requestName,

- the OBJECT IDENTIFIER (if any) assigned to the responseName (note that the same OBJECT IDENTIFIER may be used for both the requestName and responseName),

- the format of the contents of the requestValue and responseValue (if any), and

- the semantics of the operation.

4.13. IntermediateResponse Message

While the Search operation provides a mechanism to return multiple response messages for a single Search request, other operations, by nature, do not provide for multiple response messages.

The IntermediateResponse message provides a general mechanism for defining single-request/multiple-response operations in LDAP. This message is intended to be used in conjunction with the Extended operation to define new single-request/multiple-response operations or in conjunction with a control when extending existing LDAP operations in a way that requires them to return Intermediate response information.

It is intended that the definitions and descriptions of Extended operations and controls that make use of the IntermediateResponse message will define the circumstances when an IntermediateResponse message can be sent by a server and the associated meaning of an IntermediateResponse message sent in a particular circumstance.

```
IntermediateResponse ::= [APPLICATION 25] SEQUENCE {
        responseName     [0] LDAPOID OPTIONAL,
        responseValue    [1] OCTET STRING OPTIONAL }
```

IntermediateResponse messages SHALL NOT be returned to the client unless the client issues a request that specifically solicits their return. This document defines two forms of solicitation: Extended operation and request control. IntermediateResponse messages are specified in documents describing the manner in which they are solicited (i.e., in the Extended operation or request control specification that uses them). These specifications include:

- the OBJECT IDENTIFIER (if any) assigned to the responseName,

- the format of the contents of the responseValue (if any), and

- the semantics associated with the IntermediateResponse message.

Extensions that allow the return of multiple types of IntermediateResponse messages SHALL identify those types using unique responseName values (note that one of these may specify no value).

Sections 4.13.1 and 4.13.2 describe additional requirements on the inclusion of responseName and responseValue in IntermediateResponse messages.

4.13.1. Usage with LDAP ExtendedRequest and ExtendedResponse

A single-request/multiple-response operation may be defined using a single ExtendedRequest message to solicit zero or more IntermediateResponse messages of one or more kinds, followed by an ExtendedResponse message.

4.13.2. Usage with LDAP Request Controls

A control's semantics may include the return of zero or more IntermediateResponse messages prior to returning the final result code for the operation. One or more kinds of IntermediateResponse messages may be sent in response to a request control.

All IntermediateResponse messages associated with request controls SHALL include a responseName. This requirement ensures that the client can correctly identify the source of IntermediateResponse messages when:

- two or more controls using IntermediateResponse messages are included in a request for any LDAP operation or

- one or more controls using IntermediateResponse messages are included in a request with an LDAP Extended operation that uses IntermediateResponse messages.

4.14. StartTLS Operation

The Start Transport Layer Security (StartTLS) operation's purpose is to initiate installation of a TLS layer. The StartTLS operation is defined using the Extended operation mechanism described in Section 4.12.

4.14.1. StartTLS Request

A client requests TLS establishment by transmitting a StartTLS request message to the server. The StartTLS request is defined in terms of an ExtendedRequest. The requestName is "1.3.6.1.4.1.1466.20037", and the requestValue field is always absent.

The client MUST NOT send any LDAP PDUs at this LDAP message layer following this request until it receives a StartTLS Extended response and, in the case of a successful response, completes TLS negotiations.

Detected sequencing problems (particularly those detailed in Section 3.1.1 of [RFC4513]) result in the resultCode being set to operationsError.

If the server does not support TLS (whether by design or by current configuration), it returns with the resultCode set to protocolError as described in Section 4.12.

4.14.2. StartTLS Response

When a StartTLS request is received, servers supporting the operation MUST return a StartTLS response message to the requestor. The responseName is "1.3.6.1.4.1.1466.20037" when provided (see Section 4.12). The responseValue is always absent.

If the server is willing and able to negotiate TLS, it returns the StartTLS response with the resultCode set to success. Upon client receipt of a successful StartTLS response, protocol peers may commence with TLS negotiation as discussed in Section 3 of [RFC4513].

If the server is otherwise unwilling or unable to perform this operation, the server is to return an appropriate result code indicating the nature of the problem. For example, if the TLS subsystem is not presently available, the server may indicate this by returning with the resultCode set to unavailable. In cases where a non-success result code is returned, the LDAP session is left without a TLS layer.

4.14.3. Removal of the TLS Layer

Either the client or server MAY remove the TLS layer and leave the LDAP message layer intact by sending and receiving a TLS closure alert.

The initiating protocol peer sends the TLS closure alert and MUST wait until it receives a TLS closure alert from the other peer before sending further LDAP PDUs.

When a protocol peer receives the initial TLS closure alert, it may choose to allow the LDAP message layer to remain intact. In this case, it MUST immediately transmit a TLS closure alert. Following this, it MAY send and receive LDAP PDUs.

Protocol peers MAY terminate the LDAP session after sending or receiving a TLS closure alert.

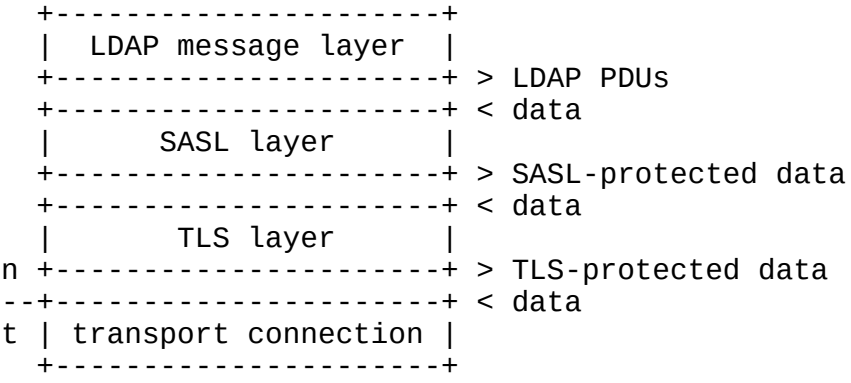
5. Protocol Encoding, Connection, and Transfer

This protocol is designed to run over connection-oriented, reliable transports, where the data stream is divided into octets (8-bit units), with each octet and each bit being significant.

One underlying service, LDAP over TCP, is defined in Section 5.2. This service is generally applicable to applications providing or consuming X.500-based directory services on the Internet. This specification was generally written with the TCP mapping in mind. Specifications detailing other mappings may encounter various obstacles.

Implementations of LDAP over TCP MUST implement the mapping as described in Section 5.2.

This table illustrates the relationship among the different layers involved in an exchange between two protocol peers:



5.1. Protocol Encoding

The protocol elements of LDAP SHALL be encoded for exchange using the Basic Encoding Rules [BER] of [ASN.1] with the following restrictions:

- Only the definite form of length encoding is used.

- OCTET STRING values are encoded in the primitive form only.

- If the value of a BOOLEAN type is true, the encoding of the value octet is set to hex "FF".

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

These restrictions are meant to ease the overhead of encoding and decoding certain elements in BER.

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

5.2. Transmission Control Protocol (TCP)

The encoded LDAPMessage PDUs are mapped directly onto the TCP [RFC793] 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 Internet Assigned Numbers Authority (IANA)-assigned LDAP port, 389 [PortReg]. Servers may instead provide a listener on a different port number. Clients MUST support contacting servers on any valid TCP port.

5.3. Termination of the LDAP session

Termination of the LDAP session is typically initiated by the client sending an UnbindRequest (Section 4.3), or by the server sending a Notice of Disconnection (Section 4.4.1). In these cases, each protocol peer gracefully terminates the LDAP session by ceasing exchanges at the LDAP message layer, tearing down any SASL layer, tearing down any TLS layer, and closing the transport connection.

A protocol peer may determine that the continuation of any communication would be pernicious, and in this case, it may abruptly terminate the session by ceasing communication and closing the transport connection.

In either case, when the LDAP session is terminated, uncompleted operations are handled as specified in Section 3.1.

6. Security Considerations

This version of the protocol provides facilities for simple authentication using a cleartext password, as well as any SASL [RFC4422] mechanism. Installing SASL and/or TLS layers can provide integrity and other data security services.

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.

Servers are encouraged to prevent directory modifications by clients that have authenticated anonymously [RFC4513].

Security considerations for authentication methods, SASL mechanisms, and TLS are described in [RFC4513].

Note that SASL authentication exchanges do not provide data confidentiality or integrity protection for the version or name fields of the BindRequest or the resultCode, diagnosticMessage, or referral fields of the BindResponse, nor for any information contained in controls attached to Bind requests or responses. Thus, information contained in these fields SHOULD NOT be relied on unless it is otherwise protected (such as by establishing protections at the transport layer).

Implementors should note that various security factors (including authentication and authorization information and data security services) may change during the course of the LDAP session or even during the performance of a particular operation. For instance, credentials could expire, authorization identities or access controls could change, or the underlying security layer(s) could be replaced or terminated. Implementations should be robust in the handling of changing security factors.

In some cases, it may be appropriate to continue the operation even in light of security factor changes. For instance, it may be appropriate to continue an Abandon operation regardless of the change, or to continue an operation when the change upgraded (or maintained) the security factor. In other cases, it may be appropriate to fail or alter the processing of the operation. For instance, if confidential protections were removed, it would be appropriate either to fail a request to return sensitive data or, minimally, to exclude the return of sensitive data.

Implementations that 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 that 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.

Servers may return referrals or Search result references that redirect clients to peer servers. It is possible for a rogue application to inject such referrals into the data stream in an attempt to redirect a client to a rogue server. Clients are advised to be aware of this and possibly reject referrals when confidentiality measures are not in place. Clients are advised to reject referrals from the StartTLS operation.

The matchedDN and diagnosticMessage fields, as well as some resultCode values (e.g., attributeOrValueExists and entryAlreadyExists), could disclose the presence or absence of specific data in the directory that is subject to access and other administrative controls. Server implementations should restrict access to protected information equally under both normal and error conditions.

Protocol peers MUST be prepared to handle invalid and arbitrary-length protocol encodings. Invalid protocol encodings include: BER encoding exceptions, format string and UTF-8 encoding exceptions, overflow exceptions, integer value exceptions, and binary mode on/off flag exceptions. The LDAPv3 PROTOS [PROTOS-LDAP] test suite provides excellent examples of these exceptions and test cases used to discover flaws.

In the event that a protocol peer senses an attack that in its nature could cause damage due to further communication at any layer in the LDAP session, the protocol peer should abruptly terminate the LDAP session as described in Section 5.3.

7. Acknowledgements

This document is based on RFC 2251 by Mark Wahl, Tim Howes, and Steve Kille. RFC 2251 was a product of the IETF ASID Working Group.

It is also based on RFC 2830 by Jeff Hodges, RL "Bob" Morgan, and Mark Wahl. RFC 2830 was a product of the IETF LDAPEXT Working Group.

It is also based on RFC 3771 by Roger Harrison and Kurt Zeilenga. RFC 3771 was an individual submission to the IETF.

This document is a product of the IETF LDAPBIS Working Group. Significant contributors of technical review and content include Kurt Zeilenga, Steven Legg, and Hallvard Furuseth.

8. Normative References

[ASN.1] ITU-T Recommendation X.680 (07/2002) | ISO/IEC 8824-1:2002 "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[BER] ITU-T Rec. X.690 (07/2002) | ISO/IEC 8825-1:2002, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", 2002.

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

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3454] Hoffman P. and M. Blanchet, "Preparation of Internationalized Strings ('stringprep')", RFC 3454, December 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", RFC 4346, March 2006.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4512] Zeilenga, K., Lightweight Directory Access Protocol (LDAP): Directory Information Models", RFC 4512, June 2006.

[RFC4513] Harrison, R., Ed., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

[RFC4514] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC4516] Smith, M., Ed. and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator", RFC 4516, June 2006.

[RFC4517] Legg, S., Ed., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.

[RFC4520] Zeilenga, K., "Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)", BCP 64, RFC 4520, June 2006.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 3.2.0" is defined by "The Unicode Standard, Version 3.0" (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the "Unicode Standard Annex #27: Unicode 3.1" (http://www.unicode.org/reports/tr27/) and by the "Unicode Standard Annex #28: Unicode 3.2" (http://www.unicode.org/reports/tr28/).

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

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

9. Informative References

[CharModel] Whistler, K. and M. Davis, "Unicode Technical Report #17, Character Encoding Model", UTR17, <http://www.unicode.org/unicode/reports/tr17/>, August 2000.

[Glossary] The Unicode Consortium, "Unicode Glossary", <http://www.unicode.org/glossary/>.

[PortReg] IANA, "Port Numbers", <http://www.iana.org/assignments/port-numbers>.

[PROTOS-LDAP] University of Oulu, "PROTOS Test-Suite: c06-ldapv3" <http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/ldapv3/>.

10. IANA Considerations

The Internet Assigned Numbers Authority (IANA) has updated the LDAP result code registry to indicate that this document provides the definitive technical specification for result codes 0-36, 48-54, 64-70, 80-90. It is also noted that one resultCode value (strongAuthRequired) has been renamed (to strongerAuthRequired).

The IANA has also updated the LDAP Protocol Mechanism registry to indicate that this document and [RFC4513] provides the definitive technical specification for the StartTLS (1.3.6.1.4.1.1466.20037) Extended operation.

IANA has assigned LDAP Object Identifier 18 [RFC4520] to identify the ASN.1 module defined in this document.

```
Subject: Request for LDAP Object Identifier Registration
Person & email address to contact for further information:
     Jim Sermersheim <jimse@novell.com>
Specification: RFC 4511
Author/Change Controller: IESG
Comments:
     Identifies the LDAP ASN.1 module
```

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 4.1.9.

Additional result codes MAY be defined for use with extensions [RFC4520]. Client implementations SHALL treat any result code that they do not recognize as an unknown error condition.

The descriptions provided here do not fully account for result code substitutions used to prevent unauthorized disclosures (such as substitution of noSuchObject for insufficientAccessRights, or invalidCredentials for insufficientAccessRights).

A.1. Non-Error Result Codes

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

```
success (0),
compareFalse (5),
compareTrue (6),
referral (10), and
saslBindInProgress (14).
```

The success, compareTrue, and compareFalse result codes indicate successful completion (and, hence, are referred to as "successful" result codes).

The referral and saslBindInProgress result codes indicate the client needs to take additional action to complete the operation.

A.2. Result Codes

Existing LDAP result codes are described as follows:

success (0)
: Indicates the successful completion of an operation. Note: this code is not used with 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 StartTLS [RFC4346] while there are other uncompleted operations or if a TLS layer was already installed.

protocolError (2)
Indicates the server received data that is not well-formed.

For Bind operation only, this code is also used to indicate that the server does not support the requested protocol version.

For Extended operations only, this code is also used to indicate that the server does not support (by design or configuration) the Extended operation associated with the requestName.

For request operations specifying multiple controls, this may be used to indicate that the server cannot ignore the order of the controls as specified, or that the combination of the specified controls is invalid or unspecified.

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 Compare operation has successfully completed and the assertion has evaluated to FALSE or Undefined.

compareTrue (6)
Indicates that the Compare operation has successfully completed and the assertion has evaluated to TRUE.

authMethodNotSupported (7)
Indicates that the authentication method or mechanism is not supported.

strongerAuthRequired (8)
Indicates the server requires strong(er) authentication in order to complete the operation.

When used with the Notice of Disconnection operation, this code indicates that the server has detected that an established security association between the client and server has unexpectedly failed or been compromised.

referral (10)
Indicates that a referral needs to be chased to complete the operation (see Section 4.1.10).

adminLimitExceeded (11)
Indicates that an administrative limit has been exceeded.

unavailableCriticalExtension (12)
Indicates a critical control is unrecognized (see Section 4.1.11).

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 unrecognized attribute description.

inappropriateMatching (18)
Indicates that an attempt was made (e.g., in an assertion) to use a matching rule not defined for the attribute type concerned.

constraintViolation (19)
Indicates that the client supplied an attribute value that does not conform to the constraints placed upon it by the data model.

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

attributeOrValueExists (20)
Indicates that the client supplied an attribute or value to be added to an entry, but the attribute or value 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. For example, the code may used to indicate an alias has been dereferenced that names no object.

invalidDNSyntax (34)
Indicates that an 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 that do not conform to the syntax of the attribute's type.

aliasDereferencingProblem (36)
Indicates that a problem occurred while dereferencing an alias. Typically, an alias was encountered in a situation where it was not allowed or where access was denied.

inappropriateAuthentication (48)
Indicates the server requires the client that had attempted to bind anonymously or without supplying credentials to provide some form of credentials.

invalidCredentials (49)
Indicates that the provided credentials (e.g., the user's name and password) 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 too busy to service the operation.

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 (e.g., while dereferencing aliases or chaining an operation).

namingViolation (64)
Indicates that the entry's name violates naming restrictions.

objectClassViolation (65)
Indicates that the entry violates object class restrictions.

notAllowedOnNonLeaf (66)
Indicates that the operation is inappropriately acting upon a non-leaf entry.

notAllowedOnRDN (67)
Indicates that the operation is inappropriately attempting to remove a value that forms the entry's relative distinguished name.

entryAlreadyExists (68)
Indicates that the request cannot be fulfilled (added, moved, or renamed) as the target entry already exists.

objectClassModsProhibited (69)
Indicates that an attempt to modify the object class(es) of an entry's 'objectClass' attribute is prohibited.

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

affectsMultipleDSAs (71)
Indicates that the operation cannot be performed as it would affect 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 {1 3 6 1 1 18}
-- Copyright (C) The Internet Society (2006).  This version of
-- this ASN.1 module is part of RFC 4511; see the RFC itself
-- for full legal notices.
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,
          ...,
          intermediateResponse  IntermediateResponse },
     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>
                         -- [RFC4512]

LDAPDN ::= LDAPString -- Constrained to <distinguishedName>
                      -- [RFC4514]

RelativeLDAPDN ::= LDAPString -- Constrained to <name-component>
                              -- [RFC4514]

AttributeDescription ::= LDAPString
                        -- Constrained to <attributedescription>
                        -- [RFC4512]

AttributeValue ::= OCTET STRING

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

AssertionValue ::= OCTET STRING

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

Attribute ::= PartialAttribute(WITH COMPONENTS {
     ...,
     vals (SIZE(1..MAX))})

MatchingRuleId ::= LDAPString

LDAPResult ::= SEQUENCE {
     resultCode         ENUMERATED {
          success                      (0),
          operationsError              (1),
          protocolError                (2),
          timeLimitExceeded            (3),
          sizeLimitExceeded            (4),
          compareFalse                 (5),
          compareTrue                  (6),
          authMethodNotSupported       (7),
          strongerAuthRequired         (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),
                ...  },
           matchedDN          LDAPDN,
           diagnosticMessage  LDAPString,
           referral           [3] Referral OPTIONAL }

      Referral ::= SEQUENCE SIZE (1..MAX) OF uri URI

      URI ::= LDAPString     -- limited to characters permitted in
                             -- URIs

      Controls ::= SEQUENCE OF control 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      AttributeSelection }

AttributeSelection ::= SEQUENCE OF selector LDAPString
               -- The LDAPString is constrained to
               -- <attributeSelector> in Section 4.5.1.8

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

     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
                          partialAttribute PartialAttribute

     SearchResultReference ::= [APPLICATION 19] SEQUENCE
                               SIZE (1..MAX) OF uri URI

     SearchResultDone ::= [APPLICATION 5] LDAPResult

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

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

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

AttributeList ::= SEQUENCE OF attribute Attribute

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,
     responseValue    [11] OCTET STRING OPTIONAL }

IntermediateResponse ::= [APPLICATION 25] SEQUENCE {
     responseName     [0] LDAPOID OPTIONAL,
     responseValue    [1] OCTET STRING OPTIONAL }

END
```

Appendix C. Changes

This appendix is non-normative.

This appendix summarizes substantive changes made to RFC 2251, RFC 2830, and RFC 3771.

C.1. Changes Made to RFC 2251

This section summarizes the substantive changes made to Sections 1, 2, 3.1, and 4, and the remainder of RFC 2251. Readers should consult [RFC4512] and [RFC4513] for summaries of changes to other sections.

C.1.1. Section 1 (Status of this Memo)

- Removed IESG note. Post publication of RFC 2251, mandatory LDAP authentication mechanisms have been standardized which are sufficient to remove this note. See [RFC4513] for authentication mechanisms.

C.1.2. Section 3.1 (Protocol Model) and others

- Removed notes giving history between LDAP v1, v2, and v3. Instead, added sufficient language so that this document can stand on its own.

C.1.3. Section 4 (Elements of Protocol)

- Clarified where the extensibility features of ASN.1 apply to the protocol. This change affected various ASN.1 types by the inclusion of ellipses (...) to certain elements.
- Removed the requirement that servers that implement version 3 or later MUST provide the 'supportedLDAPVersion' attribute. This statement provided no interoperability advantages.

C.1.4. Section 4.1.1 (Message Envelope)

- There was a mandatory requirement for the server to return a Notice of Disconnection and drop the transport connection when a PDU is malformed in a certain way. This has been updated such that the server SHOULD return the Notice of Disconnection, and it MUST terminate the LDAP Session.

C.1.5. Section 4.1.1.1 (Message ID)

- Required that the messageID of requests MUST be non-zero as the zero is reserved for Notice of Disconnection.

- Specified when it is and isn't appropriate to return an already used messageID. RFC 2251 accidentally imposed synchronous server behavior in its wording of this.

C.1.6. Section 4.1.2 (String Types)

- Stated that LDAPOID is constrained to <numericoid> from [RFC4512].

C.1.7. Section 4.1.5.1 (Binary Option) and others

- Removed the Binary Option from the specification. There are numerous interoperability problems associated with this method of alternate attribute type encoding. Work to specify a suitable replacement is ongoing.

C.1.8. Section 4.1.8 (Attribute)

- Combined the definitions of PartialAttribute and Attribute here, and defined Attribute in terms of PartialAttribute.

C.1.9. Section 4.1.10 (Result Message)

- Renamed "errorMessage" to "diagnosticMessage" as it is allowed to be sent for non-error results.
- Moved some language into Appendix A, and referred the reader there.
- Allowed matchedDN to be present for other result codes than those listed in RFC 2251.
- Renamed the code "strongAuthRequired" to "strongerAuthRequired" to clarify that this code may often be returned to indicate that a stronger authentication is needed to perform a given operation.

C.1.10. Section 4.1.11 (Referral)

- Defined referrals in terms of URIs rather than URLs.
- Removed the requirement that all referral URIs MUST be equally capable of progressing the operation. The statement was ambiguous and provided no instructions on how to carry it out.
- Added the requirement that clients MUST NOT loop between servers.
- Clarified the instructions for using LDAPURLs in referrals, and in doing so added a recommendation that the scope part be present.
- Removed imperatives which required clients to use URLs in specific ways to progress an operation. These did nothing for interoperability.

C.1.11. Section 4.1.12 (Controls)

- Specified how control values defined in terms of ASN.1 are to be encoded.
- Noted that the criticality field is only applied to request messages (except UnbindRequest), and must be ignored when present on response messages and UnbindRequest.
- Specified that non-critical controls may be ignored at the server's discretion. There was confusion in the original wording which led some to believe that recognized controls may not be ignored as long as they were associated with a proper request.
- Added language regarding combinations of controls and the ordering of controls on a message.
- Specified that when the semantics of the combination of controls is undefined or unknown, it results in a protocolError.
- Changed "The server MUST be prepared" to "Implementations MUST be prepared" in paragraph 8 to reflect that both client and server implementations must be able to handle this (as both parse controls).

C.1.12. Section 4.2 (Bind Operation)

- Mandated that servers return protocolError when the version is not supported.
- Disambiguated behavior when the simple authentication is used, the name is empty, and the password is non-empty.
- Required servers to not dereference aliases for Bind. This was added for consistency with other operations and to help ensure data consistency.
- Required that textual passwords be transferred as UTF-8 encoded Unicode, and added recommendations on string preparation. This was to help ensure interoperability of passwords being sent from different clients.

C.1.13. Section 4.2.1 (Sequencing of the Bind Request)

- This section was largely reorganized for readability, and language was added to clarify the authentication state of failed and abandoned Bind operations.
- Removed: "If a SASL transfer encryption or integrity mechanism has been negotiated, that mechanism does not support the changing of credentials from one identity to another, then the client MUST instead establish a new connection."
  If there are dependencies between multiple negotiations of a particular SASL mechanism, the technical specification for that SASL mechanism details how applications are to deal with them. LDAP should not require any special handling.
- Dropped MUST imperative in paragraph 3 to align with [RFC2119].

- Mandated that clients not send non-Bind operations while a Bind is in progress, and suggested that servers not process them if they are received. This is needed to ensure proper sequencing of the Bind in relationship to other operations.

C.1.14. Section 4.2.3 (Bind Response)

- Moved most error-related text to Appendix A, and added text regarding certain errors used in conjunction with the Bind operation.
- Prohibited the server from specifying serverSaslCreds when not appropriate.

C.1.15. Section 4.3 (Unbind Operation)

- Specified that both peers are to cease transmission and terminate the LDAP session for the Unbind operation.

C.1.16. Section 4.4 (Unsolicited Notification)

- Added instructions for future specifications of Unsolicited Notifications.

C.1.17. Section 4.5.1 (Search Request)

- SearchRequest attributes is now defined as an AttributeSelection type rather than AttributeDescriptionList, and an ABNF is provided.
- SearchRequest attributes may contain duplicate attribute descriptions. This was previously prohibited. Now servers are instructed to ignore subsequent names when they are duplicated. This was relaxed in order to allow different short names and also OIDs to be requested for an attribute.
- The present search filter now evaluates to Undefined when the specified attribute is not known to the server. It used to evaluate to FALSE, which caused behavior inconsistent with what most would expect, especially when the 'not' operator was used.
- The Filter choice SubstringFilter substrings type is now defined with a lower bound of 1.
- The SubstringFilter substrings 'initial, 'any', and 'final' types are now AssertionValue rather than LDAPString. Also, added imperatives stating that 'initial' (if present) must be listed first, and 'final' (if present) must be listed last.
- Disambiguated the semantics of the derefAliases choices. There was question as to whether derefInSearching applied to the base object in a wholeSubtree Search.
- Added instructions for equalityMatch, substrings, greaterOrEqual, lessOrEqual, and approxMatch.

C.1.18. Section 4.5.2 (Search Result)

- Recommended that servers not use attribute short names when it knows they are ambiguous or may cause interoperability problems.
- Removed all mention of ExtendedResponse due to lack of implementation.

C.1.19. Section 4.5.3 (Continuation References in the Search Result)

- Made changes similar to those made to Section 4.1.11.

C.1.20. Section 4.5.3.1 (Example)

- Fixed examples to adhere to changes made to Section 4.5.3.

C.1.21. Section 4.6 (Modify Operation)

- Replaced AttributeTypeAndValues with Attribute as they are equivalent.
- Specified the types of modification changes that might temporarily violate schema. Some readers were under the impression that any temporary schema violation was allowed.

C.1.22. Section 4.7 (Add Operation)

- Aligned Add operation with X.511 in that the attributes of the RDN are used in conjunction with the listed attributes to create the entry. Previously, Add required that the distinguished values be present in the listed attributes.
- Removed requirement that the objectClass attribute MUST be specified as some DSE types do not require this attribute. Instead, generic wording was added, requiring the added entry to adhere to the data model.
- Removed recommendation regarding placement of objects. This is covered in the data model document.

C.1.23. Section 4.9 (Modify DN Operation)

- Required servers to not dereference aliases for Modify DN. This was added for consistency with other operations and to help ensure data consistency.
- Allow Modify DN to fail when moving between naming contexts.
- Specified what happens when the attributes of the newrdn are not present on the entry.

C.1.24. Section 4.10 (Compare Operation)

- Specified that compareFalse means that the Compare took place and the result is false. There was confusion that led people to believe that an Undefined match resulted in compareFalse.
- Required servers to not dereference aliases for Compare. This was added for consistency with other operations and to help ensure data consistency.

C.1.25. Section 4.11 (Abandon Operation)

- Explained that since Abandon returns no response, clients should not use it if they need to know the outcome.
- Specified that Abandon and Unbind cannot be abandoned.

C.1.26. Section 4.12 (Extended Operation)

- Specified how values of Extended operations defined in terms of ASN.1 are to be encoded.
- Added instructions on what Extended operation specifications consist of.
- Added a recommendation that servers advertise supported Extended operations.

C.1.27. Section 5.2 (Transfer Protocols)

- Moved referral-specific instructions into referral-related sections.

C.1.28. Section 7 (Security Considerations)

- Reworded notes regarding SASL not protecting certain aspects of the LDAP Bind messages.
- Noted that Servers are encouraged to prevent directory modifications by clients that have authenticated anonymously [RFC4513].
- Added a note regarding the possibility of changes to security factors (authentication, authorization, and data confidentiality).
- Warned against following referrals that may have been injected in the data stream.
- Noted that servers should protect information equally, whether in an error condition or not, and mentioned matchedDN, diagnosticMessage, and resultCodes specifically.
- Added a note regarding malformed and long encodings.

C.1.29. Appendix A (Complete ASN.1 Definition)

- Added "EXTENSIBILITY IMPLIED" to ASN.1 definition.
- Removed AttributeType. It is not used.

C.2. Changes Made to RFC 2830

This section summarizes the substantive changes made to Sections of RFC 2830. Readers should consult [RFC4513] for summaries of changes to other sections.

C.2.1. Section 2.3 (Response other than "success")

- Removed wording indicating that referrals can be returned from StartTLS.
- Removed requirement that only a narrow set of result codes can be returned. Some result codes are required in certain scenarios, but any other may be returned if appropriate.
- Removed requirement that the ExtendedResponse.responseName MUST be present. There are circumstances where this is impossible, and requiring this is at odds with language in Section 4.12.

C.2.1. Section 4 (Closing a TLS Connection)

- Reworded most of this section to align with definitions of the LDAP protocol layers.
- Removed instructions on abrupt closure as this is covered in other areas of the document (specifically, Section 5.3)

C.3. Changes Made to RFC 3771

- Rewrote to fit into this document. In general, semantics were preserved. Supporting and background language seen as redundant due to its presence in this document was omitted.

- Specified that Intermediate responses to a request may be of different types, and one of the response types may be specified to have no response value.

Editor's Address

Jim Sermersheim
Novell, Inc.
1800 South Novell Place
Provo, Utah 84606, USA

Phone: +1 801 861-3088
EMail: jimse@novell.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).