Network Working Group
Request for Comments: 1487

W. Yeong
Performance Systems International
T. Howes
University of Michigan
S. Kille
ISODE Consortium
July 1993

X.500 Lightweight Directory Access Protocol

Status of this Memo

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

Abstract

The protocol described in this document is designed to provide access to the Directory while not incurring the resource requirements of the Directory Access Protocol (DAP). This protocol is specifically targeted at simple management applications and browser applications that provide simple read/write interactive access to the Directory, and is intended to be a complement to the DAP itself.

Key aspects of LDAP are:

- Protocol elements are carried directly over TCP or other transport, bypassing much of the session/presentation overhead.

- Many protocol data elements are encoding as ordinary strings (e.g., Distinguished Names).

- A lightweight BER encoding is used to encode all protocol elements.

1. History

The tremendous interest in X.500 [1,2] technology in the Internet has lead to efforts to reduce the high "cost of entry" associated with use of the technology, such as the Directory Assistance Service [3] and DIXIE [4]. While efforts such as these have met with success, they have been solutions based on particular implementations and as such have limited applicability. This document continues the efforts to define Directory protocol alternatives but departs from previous efforts in that it consciously avoids dependence on particular

implementations.

2. Protocol Model

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, this is accomplished by a client transmitting a protocol request describing the operation to be performed to a server, which is then responsible for performing the necessary operations on the Directory. Upon completion of the necessary operations, the server returns a response containing any results or errors to the requesting client. In keeping with the goal of easing the costs associated with use of the Directory, it is an objective of this protocol to minimize the complexity of clients so as to facilitate widespread deployment of applications capable of utilizing the Directory.

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

Consistent with the model of servers performing protocol operations on behalf of clients, it is also to be noted that protocol servers are expected to handle referrals without resorting to the return of such referrals to the client. This protocol makes no provisions for the return of referrals to clients, as the model is one of servers ensuring the performance of all necessary operations in the Directory, with only final results or errors being returned by servers to clients.

Note that this protocol can be mapped to a strict subset of the directory abstract service, so it can be cleanly provided by the DAP.

3. Mapping Onto Transport Services

This protocol is designed to run over connection-oriented, reliable transports, with all 8 bits in an octet being significant in the data stream. Specifications for two underlying services are defined here, though others are also possible.

3.1. Transmission Control Protocol (TCP)

The LDAPMessage PDUs are mapped directly onto the TCP bytestream. Server implementations running over the TCP should provide a protocol listener on port 389.

3.2. Connection Oriented Transport Service (COTS)

The connection is established. No special special use of T-Connect is made. Each LDAPMessage PDU is mapped directly onto T-Data.

4. Elements of Protocol

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,
                               searchResponse      SearchResponse,
                               modifyRequest       ModifyRequest,
                               modifyResponse      ModifyResponse,
                               addRequest          AddRequest,
                               addResponse         AddResponse,
                               delRequest          DelRequest,
                               delResponse         DelResponse,
                               modifyRDNRequest    ModifyRDNRequest,
                               modifyRDNResponse   ModifyRDNResponse,
                               compareDNRequest    CompareRequest,
                               compareDNResponse   CompareResponse,
                               abandonRequest      AbandonRequest
                          }
      }

  MessageID ::= INTEGER (0 .. MaxInt)
```

The function of the LDAPMessage is to provide an envelope containing common fields required in all protocol exchanges. At this time the only common field is a message ID, which is required to have a value different from the values of any other requests outstanding in the LDAP session of which this message is a part.

The message ID value must be echoed in all LDAPMessage envelopes encapsulting responses corresponding to the request contained in the LDAPMessage in which the message ID value was originally used.

In addition to the LDAPMessage defined above, the following definitions are also used in defining protocol operations:

```
  IA5String ::= OCTET STRING
```

The IA5String is a notational convenience to indicate that, although strings of IA5String type encode as OCTET STRING types, the legal character set in such strings is limited to the IA5 character set.

```
  LDAPDN ::= IA5String

  RelativeLDAPDN ::= IA5String
```

An LDAPDN and a RelativeLDAPDN are respectively defined to be the representation of a Distinguished Name and a Relative Distinguished Name after encoding according to the specification in [5], such that

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

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

where <name> and <name-component> are as defined in [5].

```
  AttributeValueAssertion ::=
      SEQUENCE {
           attributeType       AttributeType
           attributeValue      AttributeValue
      }
```

The AttributeValueAssertion type definition is similar to the one in the Directory standards.

```
  AttributeType ::= IA5String

  AttributeValue ::= OCTET STRING
```

An AttributeType value takes on as its value the textual string associated with that AttributeType in the Directory standards. For example, the AttributeType ’organizationName’ with object identifier 2.5.4.10 is represented as an AttributeType in this protocol by the string "organizationName". In the event that a protocol implementation encounters an Attribute Type with which it cannot associate a textual string, an ASCII string encoding of the object identifier associated with the Attribute Type may be subsitituted. For example, the organizationName AttributeType may be represented by the ASCII string "2.5.4.10" if a protocol implementation is unable to associate the string "organizationName" with it.

A field of type AttributeValue takes on as its value an octet string encoding of a Directory AttributeValue type. The definition of these string encodings for different Directory AttributeValue types may be

found in companions to this document that define the encodings of various attribute syntaxes such as [6].

```
  LDAPResult ::=
      SEQUENCE {
          resultCode    ENUMERATED {
                          success                      (0),
                          operationsError              (1),
                          protocolError                (2),
                          timeLimitExceeded            (3),
                          sizeLimitExceeded            (4),
                          compareFalse                 (5),
                          compareTrue                  (6),
                          authMethodNotSupported       (7),
                          strongAuthRequired           (8),
                          noSuchAttribute              (16),
                          undefinedAttributeType       (17),
                          inappropriateMatching        (18),
                          constraintViolation          (19),
                          attributeOrValueExists       (20),
                          invalidAttributeSyntax       (21),
                          noSuchObject                 (32),
                          aliasProblem                 (33),
                          invalidDNSyntax              (34),
                          isLeaf                       (35),
                          aliasDereferencingProblem    (36),
                          inappropriateAuthentication  (48),
                          invalidCredentials           (49),
                          insufficientAccessRights     (50),
                          busy                         (51),
                          unavailable                  (52),
                          unwillingToPerform           (53),
                          loopDetect                   (54),
                          namingViolation              (64),
                          objectClassViolation         (65),
                          notAllowedOnNonLeaf          (66),
                          notAllowedOnRDN              (67),
                          entryAlreadyExists           (68),
                          objectClassModsProhibited    (69),
                          other                        (80)
                        },
          matchedDN     LDAPDN,
          errorMessage  IA5String
      }
```

The LDAPResult is the construct used in this protocol to return success or failure indications from servers to clients. In response to various requests, servers will return responses containing fields

of type LDAPResult to indicate the final status of a protocol operation request. The errorMessage field of this construct may, at the servers option, be used to return an ASCII string containing a textual, human-readable error diagnostic. As this error diagnostic is not standardized, implementations should not rely on the values returned. If the server chooses not to return a textual diagnostic, the errorMessage field of the LDAPResult type should contain a zero length string.

For resultCodes of noSuchObject, aliasProblem, invalidDNSyntax, isLeaf, and aliasDereferencingProblem, the matchedDN field is set to the name of the lowest entry (object or alias) in the DIT that was matched and is a truncated form of the name provided or, if an alias has been dereferenced, of the resulting name. The matchedDN field should be set to NULL DN (a zero length string) in all other cases.

4.1. Bind Operation

The function of the Bind Operation is to initiate a protocol session between a client and a server, and to allow the authentication of the client to the server. The Bind Operation must be the first operation request received by a server from a client in a protocol session. The Bind Request is defined as follows:

```
BindRequest ::=
    [APPLICATION 0] SEQUENCE {
                        version   INTEGER (1 .. 127),
                        name      LDAPDN,
                        authentication CHOICE {
                             simple        [0] OCTET STRING,
                             krbv42LDAP    [1] OCTET STRING,
                             krbv42DSA     [2] OCTET STRING
                        }
    }
```

Parameters of the Bind Request are:

- version: A version number indicating the version of the protocol to be used in this protocol session. This document describes version 2 of the LDAP protocol. Note that there is no version negotiation, and the client should just set this parameter to the version it desires.

- name: The name of the Directory object that the client wishes to bind as. This field may take on a null value (a zero length string) for the purposes of anonymous binds.

- authentication: information used to authenticate the name, if any,

provided in the Bind Request. The "simple" authentication option provides minimal authentication facilities, with the contents of the authentication field consisting only of a cleartext password. This option should also be used when unauthenticated or anonymous binds are to be performed, with the field containing a zero length string in such cases. Kerberos version 4 [7] authentication to the LDAP server and the DSA is accomplished by using the "krbv42LDAP" and "krbv42DSA" authentication options, respectively. Note that though they are referred to as separate entities here, there is no requirement these two entities be distinct (i.e., a DSA could speak LDAP directly). Two separate authentication options are provided to support all implementations. Each octet string should contain the kerberos ticket (e.g., as returned by krb_mk_req()) for the appropriate service. The suggested service name for authentication to the LDAP server is "ldapserver". The suggested service name for authentication to the DSA is "x500dsa". In both cases, the suggested instance name for the service is the name of the host on which the service is running. Of course, the actual service names and instances will depend on what is entered in the local kerberos principle database.

The Bind Operation requires a response, the Bind Response, which is defined as:

```
BindResponse ::= [APPLICATION 1] LDAPResult
```

A Bind Response consists simply of an indication from the server of the status of the client's request for the initiation of a protocol session.

Upon receipt of a Bind Request, a protocol server will authenticate the requesting client if necessary, and attempt to set up a protocol session with that client. The server will then return a Bind Response to the client indicating the status of the session setup request.

4.2. Unbind Operation

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

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

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

4.3. Search Operation

The Search Operation allows a client to request that a search be performed on its behalf by a server. 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),
        attrsOnly     BOOLEAN,
        filter        Filter,
        attributes    SEQUENCE OF AttributeType
}

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

SubstringFilter
    SEQUENCE {
        type              AttributeType,
        SEQUENCE OF CHOICE {
            initial       [0] IA5String,
            any           [1] IA5String,
            final         [2] IA5String
        }
```

```
        }
```

Parameters of the Search Request are:

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

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

- derefAliases: An indicator as to how alias objects should be handled in searching. The semantics of the possible values of this field are, in order of increasing value:

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

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

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

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

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

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

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

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

- attributes: A list of the attributes from each entry found as a

result of the search to be returned. An empty list signifies that all attributes from each entry found in the search are to be returned.

The results of the search attempted by the server upon receipt of a Search Request are returned in Search Responses, defined as follows:

```
Search Response ::=
    CHOICE {
         entry          [APPLICATION 4] SEQUENCE {
                              objectName     LDAPDN,
                              attributes     SEQUENCE OF SEQUENCE {
                                                  AttributeType,
                                                  SET OF AttributeValue
                                             }
                         },
         resultCode     [APPLICATION 5] LDAPResult
     }
```

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

The server will return to the client a sequence of responses comprised of:

- Zero or more Search Responses each consisting of an entry found during the search; with the response sequence terminated by

- A single Search Response containing an indication of success, or detailing any errors that have occurred.

Each entry returned will contain all attributes, complete with associated values if necessary, as specified in the 'attributes' field of the Search Request.

Note that an X.500 "list" operation can be emulated by a one-level LDAP search operation with a filter checking for the existence of the objectClass attribute, and that an X.500 "read" operation can be emulated by a base object LDAP search operation with the same filter.

4.4. Modify Operation

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

```
ModifyRequest ::=
    [APPLICATION 6] SEQUENCE {
         object         LDAPDN,
         modification   SEQUENCE OF SEQUENCE {
                              operation      ENUMERATED {
                                                  add       (0),
                                                  delete    (1),
                                                  replace   (2)
                                             },
                              modification   SEQUENCE {
                                                 type    AttributeType,
                                                 values  SET OF
                                                           AttributeValue
                                             }
                         }
    }
```

Parameters of the Modify Request are:

- object: The object to be modified. The value of this field should name the object to be modified after all aliases have been dereferenced. The server will not perform any alias dereferencing in determining the object to be modified.

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

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

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

  replace: replace existing values of the given attribute with the new values listed, creating the attribute if necessary.

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

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

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

The server will return to the client a single Modify Response indicating either the successful completion of the DIB modification, or the reason that the modification failed. Note that due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications of the DIB 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.

4.5. 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,
         attrs          SEQUENCE OF SEQUENCE {
                              type          AttributeType,
                              values        SET OF AttributeValue
                        }
    }
```

Parameters of the Add Request are:

- entry: the Distinguished Name of the entry to be added. Note that all components of the name except for the last RDN component must exist for the add to succeed.

- attrs: the list of attributes that make up the content of the entry being added.

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

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

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

4.6. 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 only of the Distinguished Name of the entry to be deleted. The result of the delete attempted by the server upon receipt of a Delete Request is returned in the Delete Response, defined as follows:

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

Upon receipt of a Delete Request, a server will attempt to perform the entry removal requested. The result of the delete attempt will be returned to the client in the Delete Response. Note that only leaf objects may be deleted with this operation.

4.7. Modify RDN Operation

The Modify RDN Operation allows a client to change the last component of the name of an entry in the Directory. The Modify RDN Request is defined as follows:

```
ModifyRDNRequest ::=
    [APPLICATION 12] SEQUENCE {
         entry          LDAPDN,
         newrdn         RelativeLDAPDN
    }
```

Parameters of the Modify RDN Request are:

- entry: the name of the entry to be changed.

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

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

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

Upon receipt of a Modify RDN Request, a server will attempt to perform the name change. The result of the name change attempt will be returned to the client in the Modify RDN Response. The attributes that make up the old RDN are deleted from the entry.

4.8. Compare Operation

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

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

Parameters of the Compare Request are:

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

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

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

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

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

4.9. Abandon Operation

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

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

There is no response defined in the Abandon Operation. Upon transmission of an Abandon Operation, a client may expect that the operation identified by the Message ID in the Abandon Request has been abandoned. In the event that a server receives an Abandon Request on a Search Operation in the midst of transmitting responses to that search, that server should cease transmitting responses to the abandoned search immediately.

5. Protocol Element Encodings

The protocol elements of LDAP are encoded for exchange using the Basic Encoding Rules (BER) [11] of ASN.1 [10]. However, due to the high overhead involved in using certain elements of the BER, the following additional restrictions are placed on BER-encodings of LDAP protocol elements:

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

(2) Bitstrings and octet strings will be encoded in the primitive form only.

6. Security Considerations

This version of the protocol provides facilities only for simple authentication using a cleartext password, and for kerberos version 4 authentication. Future versions of LDAP will likely include support for other authentication methods.

7. Bibliography

[1] The Directory: Overview of Concepts, Models and Service. CCITT Recommendation X.500, 1988.

[2] Information Processing Systems -- Open Systems Interconnection -- The Directory: Overview of Concepts, Models and Service. ISO/IEC JTC 1/SC21; International Standard 9594-1, 1988.

[3] Rose, M., "Directory Assistance Service", RFC 1202, Performance Systems International, Inc., February 1991.

[4] Howes, R., Smith, M., and B. Beecher, "DIXIE Protocol Specification", RFC 1249, University of Michigan, August 1991.

[5] Kille, S., "A String Representation of Distinguished Names", RFC 1485, ISODE Consortium, July 1993.

[6] Howes, T., Kille, S., Yeong, W., and C. Robbins, "The String Representation of Standard Attribute Syntaxes", RFC 1488, University of Michigan, ISODE Consortium, Performance Systems International, NeXor Ltd., July 1993.

[7] Kerberos Authentication and Authorization System. S.P. Miller, B.C. Neuman, J.I. Schiller, J.H. Saltzer; MIT Project Athena Documentation Section E.2.1, December 1987.

[8] The Directory: Models. CCITT Recommendation X.501 ISO/IEC JTC 1/SC21; International Standard 9594-2, 1988.

[9] The Directory: Abstract Service Definition. CCITT Recommendation X.511, ISO/IEC JTC 1/SC21; International Standard 9594-3, 1988.

[10] Specification of Abstract Syntax Notation One (ASN.1). CCITT Recommendation X.208, 1988.

[11] Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). CCITT Recommendation X.209, 1988.

9. Security Considerations

Security issues are not discussed in this memo.

9. Authors' Addresses

Wengyik Yeong
PSI, Inc.
510 Huntmar Park Drive
Herndon, VA 22070
USA

Phone: +1 703-450-8001
EMail: yeongw@psilink.com

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

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

Steve Kille
ISODE Consortium
PO Box 505
London
SW11 1DX
UK

Phone: +44-71-223-4062
EMail: S.Kille@isode.com

Appendix A

Complete ASN.1 Definition

```
Lightweight-Directory-Access-Protocol DEFINITIONS ::=

IMPLICIT TAGS

BEGIN

LDAPMessage ::=
    SEQUENCE {
         messageID      MessageID,
                        -- unique id in request,
                        -- to be echoed in response(s)
         protocolOp     CHOICE {
                             searchRequest       SearchRequest,
                             searchResponse      SearchResponse,
                             modifyRequest       ModifyRequest,
                             modifyResponse      ModifyResponse,
                             addRequest          AddRequest,
                             addResponse         AddResponse,
                             delRequest          DelRequest,
                             delResponse         DelResponse,
                             modifyDNRequest     ModifyDNRequest,
                             modifyDNResponse    ModifyDNResponse,
                             compareDNRequest    CompareRequest,
                             compareDNResponse   CompareResponse,
                             bindRequest         BindRequest,
                             bindResponse        BindResponse,
                             abandonRequest      AbandonRequest,
                             unbindRequest       UnbindRequest
                        }
    }

BindRequest ::=
    [APPLICATION 0] SEQUENCE {
         version        INTEGER (1 .. 127),
                        -- current version is 2
         name           LDAPDN,
                        -- null name implies an anonymous bind
         authentication CHOICE {
                             simple        [0] OCTET STRING,
                                       -- a zero length octet string
                                       -- implies an unauthenticated
                                       -- bind.
                             krbv42LDAP    [1] OCTET STRING,
                             krbv42DSA     [2] OCTET STRING
```

```
                                    -- values as returned by
                                    -- krb_mk_req()
                                    -- Other values in later
                                    -- versions of this protocol.
                        }
    }

BindResponse ::= [APPLICATION 1] LDAPResult

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),
                             alwaysDerefAliases    (3)
                        },
         sizeLimit      INTEGER (0 .. MaxInt),
                        -- value of 0 implies no sizelimit
         timeLimit      INTEGER (0 .. MaxInt),
                        -- value of 0 implies no timelimit
         attrsOnly     BOOLEAN,
                        -- TRUE, if only attributes (without values)
                        -- to be returned.
         filter         Filter,
         attributes     SEQUENCE OF AttributeType
    }

SearchResponse ::=
    CHOICE {
         entry          [APPLICATION 4] SEQUENCE {
                             objectName     LDAPDN,
                             attributes     SEQUENCE OF SEQUENCE {
                                              AttributeType,
                                              SET OF
                                                AttributeValue
                                            }
                        },
         resultCode     [APPLICATION 5] LDAPResult
    }
```

```
ModifyRequest ::=
    [APPLICATION 6] SEQUENCE {
         object         LDAPDN,
         modifications  SEQUENCE OF SEQUENCE {
                              operation     ENUMERATED {
                                              add      (0),
                                              delete   (1),
                                              replace  (2)
                                            },
                              modification  SEQUENCE {
                                              type     AttributeType,
                                              values   SET OF
                                                         AttributeValue
                                            }
                            }
    }


ModifyResponse ::= [APPLICATION 7] LDAPResult

AddRequest ::=
    [APPLICATION 8] SEQUENCE {
         entry          LDAPDN,
         attrs          SEQUENCE OF SEQUENCE {
                              type          AttributeType,
                              values        SET OF AttributeValue
                            }
    }

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyRDNRequest ::=
    [APPLICATION 12] SEQUENCE {
         entry          LDAPDN,
         newrdn         RelativeLDAPDN -- old RDN always deleted
    }

ModifyRDNResponse ::= [APPLICATION 13] LDAPResult

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

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

AbandonRequest ::= [APPLICATION 16] MessageID

MessageID ::= INTEGER (0 .. MaxInt)

LDAPDN ::= IA5String

RelativeLDAPDN ::= IA5String

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

LDAPResult ::=
    SEQUENCE {
        resultCode    ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),
                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                        noSuchAttribute              (16),
                        undefinedAttributeType       (17),
                        inappropriateMatching        (18),
                        constraintViolation          (19),
                        attributeOrValueExists       (20),
                        invalidAttributeSyntax       (21),
                        noSuchObject                 (32),
                        aliasProblem                 (33),
                        invalidDNSyntax              (34),
                        isLeaf                       (35),
                        aliasDereferencingProblem    (36),
                        inappropriateAuthentication  (48),
                        invalidCredentials           (49),
```

```
                        insufficientAccessRights     (50),
                        busy                         (51),
                        unavailable                  (52),
                        unwillingToPerform           (53),
                        loopDetect                   (54),
                        namingViolation              (64),
                        objectClassViolation         (65),
                        notAllowedOnNonLeaf          (66),
                        notAllowedOnRDN              (67),
                        entryAlreadyExists           (68),
                        objectClassModsProhibited    (69),
                        other                        (80)
                      },
        matchedDN     LDAPDN,
        errorMessage  IA5String
    }

AttributeType ::= IA5String
                -- text name of the attribute, or dotted
                -- OID representation

AttributeValue ::= OCTET STRING

AttributeValueAssertion ::=
    SEQUENCE {
        attributeType        AttributeType,
        attributeValue       AttributeValue
    }

SubstringFilter
    SEQUENCE {
        type               AttributeType,
        SEQUENCE OF CHOICE {
          initial          [0] IA5String,
          any              [1] IA5String,
          final            [2] IA5String
      }
    }

IA5String ::= OCTET STRING

MaxInt ::= 65535
END
```