Network Working Group
INTERNET-DRAFT
Obsoletes: RFC 1777, RFC 1798

M. Wahl
ISODE Consortium
T. Howes
Netscape Communications Corp.
S. Kille
ISODE Consortium

Expires in six months from 5 June 1996
Intended Category: Standards Track

Lightweight Directory Access Protocol (v3)
<draft-ietf-asid-ldapv3-protocol-01.txt>

1. Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

2. Abstract

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

Key aspects of this version of LDAP are:

- All protocol elements of LDAP (RFC 1777) and CLDAP (RFC 1798) are supported.

- Protocol elements are carried directly over TCP or other transport, bypassing much of the session/presentation overhead. Connectionless transport (UDP) is also supported for efficient lookup operations.

- Most protocol data elements can be encoded as ordinary strings

(e.g., Distinguished Names).

- Important features of X.500(1993) are included.

- Referrals to other servers may be returned.

- The protocol may be extended to support bilaterally-defined operations.

- Several service controls may be requested by the client.

3. Models

Interest in X.500 [1] directory technologies in the Internet has lead to efforts to reduce the high "cost of entry" associated with use of these technologies. This document continues the efforts to define directory protocol alternatives: it updates the LDAP [2] protocol specification, adding support for new features, including some support for servers connecting to X.500(1993).

3.1. Protocol Model

The general model adopted by this protocol is one of clients performing protocol operations against servers. In this model, a client transmits a protocol request describing the operation to be performed to a server, which is then responsible for performing the necessary operations on the directory. Upon completion of the 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 clients or servers. Rrequests and responses for multiple operations may be exchanged between a client and server in any order, provided the client eventually receives a response for every request that requires one.

In LDAP versions 1 and 2, no provision was made for protocol servers returning referrals to clients. However, for improved performance and distribution this version of the protocol permits servers to return to clients referrals to other servers if requested.
Clients may also request that no referrals be returned, in which case the server must ensure that the operation is performed against the directory, or else return an error.

Note that this protocol can be mapped to a strict subset of the X.500(1993) directory abstract service, so it can be cleanly provided by the DAP. However there is not a one-to-one mapping between LDAP protocol operations and DAP operations: some server implementations acting as a gateway to X.500 directories may need to make multiple DAP requests to perform extended operations.

3.2. Data Model

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

The LDAP protocol assumes there is one or more servers which jointly provide access to a Directory Information Tree. The tree is made up of entries. Entries have names: one or more values from the entry itself form its relative distinguished name, which must be unique among all its siblings. The concatenation of the relative distinguished names of entries, starting from the immediate subordinate of the unnamed root of the tree and continuing to a specific entry forms that entry's Distinguished Name, which is unique in the tree. An example of a Distinguished Name is

CN=Steve Kille, O=ISODE Consortium, C=GB

Entries consist of a set of attributes. An attribute is a type with one or more associated values. The attribute type, described by an OID (object identifier), governs the maximum number of values permissible for an attribute of that type in an entry, and the syntax to which the values must conform. An example of an attribute is "mail". There may be one or more values of this attribute, and they must be IA5 strings.

All the attributes of an entry are mastered together in a single server. Shadow or cached copies of entries may be held in other servers, but these cannot be updated directly by users.

3.2.1 Attributes of Entries

Each entry must have an objectClass attribute. Values of this attribute may be modified by clients, but the objectClass attribute cannot be removed. The objectClass attribute specifies the object classes of an entry, which along with the system and user schema determine the permitted attributes of an entry.

Servers should not permit clients to add attributes to an entry unless those attributes are permitted by the object class definitions, the user schema controlling that entry (specified in the subschema subentry), or are operational attributes known to that server and used for administrative purposes.

Entries may contain the following operational attributes, but if present should not be modifiable by clients:

- creatorsName: the string representation of the Distinguished Name of the user who added this entry to the directory, if known.
- createTimestamp: the GeneralizedTime value of the time this entry was added to the directory, if known.
- modifiersName: the string representation of the Distinguished Name of the user who last modified this entry, if known.
- modifyTimestamp: the GeneralizedTime value of the time this entry was last modified, if known.
- subschemaSubentry: the string representation of the Distinguished Name of the subschema subentry which controls the schema for this entry.
- entryName;binary: a DER encoding of the Distinguished Name of the entry.

Servers may implement other operational attributes. Servers which also make use of X.500(1993) protocols should provide support for the attributes defined in X.501, including administrativeRole, collectiveExclusions, governingStructureRule, dseType and entryACI.

3.2.2 Subschema Subentry

A server may provide access to one or more subschema subentries to permit clients to interrogate the schema which is in force for entries in the directory.

A server which masters or shadows entries and permits clients to modify these entries must implement subschema subentries.

The following two attributes must be present in a subschema subentries:

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

Other attributes may be present in subschema subentries.

These attributes of subschema subentries may be retrieved by requesting a baseObject search of their name, with filter set to a test for the presence of the objectClass attribute. Clients should not expect that subschema subentries will be returned in searches with other settings of scope or filter.

3.3. Relationship to X.500

This document defines LDAP in terms of X.500 as an X.500 access mechanism. An LDAP server should act in accordance with the X.500(1993) series of ITU Recommendations when providing the service.

However, it is not required that an LDAP server make use of any X.500 protocols in providing this service, e.g. LDAP can be mapped onto any other directory system so long as the X.500 data and service model is supported in the LDAP interface.

3.4. Server-specific Data Requirements

An LDAP server must provide information about itself and other information that is specific to each server. This is represented as a number of attributes located in the root DSE (DSA-Specific Entry), that which is named with the zero-length LDAPDN. These attributes should be retrievable if a client performs a base object search of the root, however they are subject to access control restrictions. They should not be included if the client performs a subtree search starting from the root. The server need not allow the client to modify these attributes.

Additional attributes may be defined by later documents or by bilateral agreement. These attributes are currently defined:

- administratorAddress: This attribute's values are string containing the addresses of the LDAP server's human administrator. This information may be of use when tracking down problems in an Internet distributed directory. For simplicity the syntax of the values are limited to being URLs of the mailto form with an RFC 822 address: "mailto:user@domain". Future versions of this protocol may permit other forms of addresses.

- currentTime: This attribute has a single value, a string containing a GeneralizedTime character string. This attribute need only be present if the server supports LDAP strong or protected simple authentication. Otherwise if the server does not know the current time, or does not choose to present it to clients, this attribute need not be present. The client may wish to use this value to detect whether a strong or protected bind is failing because the client and server clocks are not sufficiently synchronized. Clients should not use this time field for setting their own system clock.

- serverName;binary: This attribute's value is the binary representation of the ASN.1 DER encoding of the server's Distinguished Name. If the server does not have a Distinguished Name it will not be able to accept strong authentication, and this attribute should be absent. However the presence of this attribute does not guarantee that the server will be able to perform strong authentication. If the server acts as a gateway to more than one X.500 DSA capable of strong authentication, there may be multiple values of this attribute, one per DSA.

- certificationPath;binary: This attribute contains a binary DER encoding of an AF.CertificationPath data type, which is the certificate path for a server. If the server does not have a certificate path this attribute should be absent.

- namingContexts: The values of this attribute are the string representations of Distinguished Names. Each value corresponds to a naming context which this server masters or shadows. If the server does not master any information (e.g. it is an LDAP gateway to a public X.500 directory) this attribute should be absent. If the server believes it contains the entire directory, the attribute should have a single value, and that value should be the empty string (indicating the null DN of the root).

- subschemaSubentry: The values of this attribute are the string representations of Distinguished Names. Each value corresponds to a subschema subentry, which is an entry in which the server makes available attributes specifying the schema. (This is described in section 3.2.2). If the server does not master or shadow entries and does not know the locations of schema information, this attribute should be absent. If the server holds all the directory under a single set of schema rules, there will be one attribute value (and a single subentry, which could be located anywhere in the directory hierarchy). If the server holds master or shadow copies of directory entries under one or more schema rules, there may be any number of values of this attribute.

- altLdapServer: The values of this attribute are URLs of other LDAP servers which may be contacted when this server becomes unavailable. If the server does not know of any other LDAP servers which could be used this attribute should be absent. Clients should cache this information in case their preferred LDAP server later becomes unavailable.

- altX500Server: The values of this attribute are encoded with the AccessPoint93 syntax. They are the access points of X.500 DSAs which could be contacted when this server becomes unavailable. If this server does not know of any X.500 DSAs this attribute should be absent. Servers implemented as LDAP gateways to X.500 DAP may permit management clients to modify the values of this attributes.

- supportedExtension: The values of this attribute are the string representations of OBJECT IDENTIFIERs, in the dotted decimal form. Each value is the name of an extended request which this server supports (see section 4.11). If the server does not support any extended operations this attribute should be absent.

The ASN.1 type DistinguishedName is defined in [6], and the type CertificationPath is defined in [12].

4. Elements of Protocol

The LDAP protocol is described using Abstract Syntax Notation 1 [3]. It is typically transferred using a subset of the Basic Encoding Rules. In order to support future extensions to this protocol, clients and

servers should ignore elements of SEQUENCEs whose tags they do not recognize. Note that unlike X.500, each change to the LDAP protocol will have a different version number.

4.1. Common Elements

This section describes the LDAPMessage envelope 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,
                cldapUserName   LDAPDN OPTIONAL,
                protocolOp      CHOICE {
                        bindRequest     BindRequest,
                        bindReqContinue BindContinuationRequest,
                        bindRespBasic   BindResponseBasic,
                        bindRespExtd    BindResponseExtended,
                        unbindRequest   UnbindRequest,
                        searchRequest   SearchRequest,
                        searchResEntry  SearchResultEntry,
                        searchResDone   SearchResultDone,
                        searchResRef    SearchResultReference,
                        searchResFull   SearchResultFull,
                        modifyRequest   ModifyRequest,
                        modifyResponse  ModifyResponse,
                        addRequest      AddRequest,
                        addResponse     AddResponse,
                        delRequest      DelRequest,
                        delResponse     DelResponse,
                        modDNRequest    ModifyDNRequest,
                        modDNResponse   ModifyDNResponse,
                        compareRequest  CompareRequest,
                        compareResponse CompareResponse,
                        abandonRequest  AbandonRequest,
                        resumeRequest   ResumeRequest,
                        resumeError     ResumeError,
                        extendedReq     ExtendedRequest,
                        extendedResp    ExtendedResponse } }

        MessageID ::= INTEGER (0 .. maxInt)

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

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

The message ID 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. Typically a client may increment a counter for each request. The message ID value must be echoed in all LDAPMessage envelopes encapsulating responses corresponding to the request contained in the LDAPMessage in which the message ID value was originally used.

The cldapUserName identifies the requesting user for this message. It is only present if this LDAPMessage is carried in a connectionless transport protocol, such as UDP. This is described in section 5.1.3. When the LDAP session is carried in a connection-oriented transport protocol this field must be absent.

4.1.2. String Types

The LDAPString is a notational convenience to indicate that, although strings of LDAPString type encode as OCTET STRING types, the legal character set in such strings is defined to be the Basic Multilingual Plane (BMP) of UCS. These are encoded following the UTF-8 algorithm. Note that in the UTF-8 algorithm, characters which are the same as printable ASCII (0020 through 007F) are represented as that same ASCII character in a single byte.


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

The LDAPOID is a notational convenince to indicate that the permitted value of this string is a dotted-decimal representation of an OBJECT IDENTIFIER.

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

4.1.3. Distinguished Name and Relative Distinguished Name

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

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

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

where <name> and <name-component> are as defined in [4]. Only the single line representation should be used, with comma as component separator.

```
LDAPDN ::= LDAPString
```

```
RelativeLDAPDN ::= LDAPString
```

4.1.4. Attribute Type and Description

An AttributeType takes on as its value the textual string associated with that AttributeType in its specification. This string must begin with a letter, and only contain ASCII letters and digit characters. If this string is not known, the AttributeType should take the ASCII representation of its OBJECT IDENTIFIER, as decimal digits with components separated by periods, e.g., "2.5.4.10". The attribute type strings which must be supported are described in section [5].

```
AttributeType ::= LDAPString
```

An AttributeDescription is a superset of the definition of the AttributeType. It has the same ASN.1 definition, but allows additional parameters to be

```
AttributeDescription ::= LDAPString
```

A value of AttributeDescription is based on the following BNF:

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

<options> ::= <binary-option>

<binary-option> ::= "binary"
```

If the "binary" option is present, this overrides any printable encoding representation defined for that attribute. Instead the attribute is to be transferred as a BER-encoded binary value.

The data type "AttributeDescriptionList" describes a list of 0 or more attribute types. Clients and servers should be prepared to accept a list of many hundreds of attribute types.

```
AttributeDescriptionList ::= SEQUENCE SIZE (0..maxInt) OF
        AttributeDescription
```


4.1.5. Attribute Value

A field of type AttributeValue takes on as its value an octet string encoding of a X.500 Directory AttributeValue type. The definition of these string encodings for different syntaxes and types may be found in companions to this document, such as [5].

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

Note that there is no defined limit on the size of this encoding; thus PDUs including multi-megabyte attributes (e.g. photographs) may be returned. If the client has limited memory or storage capabilities it may wish to set the attrSizeLimit field when invoking a search operation.

4.1.6. Attribute Value Assertion

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

```
AttributeValueAssertion ::= SEQUENCE {
        attributeType   AttributeType,
        assertionValue  AssertionValue }

AssertionValue ::= OCTET STRING
```

For all the attributes described in [5], the assertion value syntax is the same as the value syntax.

4.1.7. Attribute

An attribute consists of a type and one or more values of that type.

```
Attribute ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET SIZE (1..maxInt) OF AttributeValue }
```

4.1.8. Matching Rule Identifier

An X.501(1993) Matching Rule is identified in the LDAP protocol by the ASCII representation of its OBJECT IDENTIFIER, as decimal digits with components separated by periods, e.g. "1.3.6.1.4.1.453.33.33".

```
MatchingRuleId ::= LDAPOID
```

4.1.9. Result Message

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

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

```
                        compareTrue                 (6),
```



```
                        authMethodNotSupported      (7),
                        strongAuthRequired          (8),
                                -- 9 reserved --
                        referral                    (10),  -- new
                        adminLimitExceeded          (11),  -- new
                        unavailableCriticalExtension (12),  -- new
                        unableToProceed             (13),  -- new
                                -- 14-15 unused --
                        noSuchAttribute             (16),
                        undefinedAttributeType      (17),
                        inappropriateMatching       (18),
                        constraintViolation         (19),
                        attributeOrValueExists      (20),
                        invalidAttributeSyntax      (21),
                        overSpecifiedFilter         (22),  -- new
                                -- 23-31 unused --
                        noSuchObject                (32),
                        aliasProblem                (33),
                        invalidDNSyntax             (34),
                        isLeaf                      (35),
                        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),
                        resultsTooLarge             (70), -- cl only
                        affectsMultipleDSAs         (71), -- new
                                -- 72-79 unused --
                        other                       (80) },
                matchedDN       LDAPDN,
                errorMessage    LDAPString,
                referral        [3] Referral OPTIONAL,
                matchedSubtype  [4] AttributeType OPTIONAL }
```

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 in a Search or Compare result. The matchedDN field should be set to a NULL DN (a zero length string) in all other cases.

When the resultCode is compareTrue the matchedSubtype field may contain the type name of the attribute whose value matched the ava in the Compare operation.

4.1.10. Referral

The referral field is present in an LDAPResult if the LDAPResult.resultCode field value is referral. It contains a reference to another server (or set of servers) which may be accessed via LDAP or other protocols.

```
Referral ::= SEQUENCE SIZE (1..maxInt) OF LDAPURL
```

The referral contains a list of URLs [14] of servers, any of which the client could contact to continue the request. Each server must be capable of processing the operation and presenting a consistent view to the client. URLs for servers implementing the LDAP protocol are written according to [9]; other kinds of URLs may be returned so long as the same information could be received using other protocols.

```
LDAPURL ::= LDAPString
```

If the server has not information of where to progress the operation that it could return to the client, it should not return a referral, but instead return the result code unableToProceed.

4.2. Bind Operation

The function of the Bind Operation is to allow authentication information to be exchanged between the client and server, and optionally allow session-wide parameters to be set.

The Bind Request is defined as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
```

```
            name                    LDAPDN,
            authentication          AuthenticationChoice,
            serviceControls         [7] Controls OPTIONAL }

    AuthenticationChoice ::= CHOICE {
            simple                  [1] OCTET STRING,
                                     -- 2 and 3 reserved
            protected               [4] ProtectedPassword,
            strong                  [5] StrongCredentials,
            otherkind               [6] OtherCredentials }

    ProtectedPassword ::= SEQUENCE {
            time1                   [0] UTCTime OPTIONAL,
            time2                   [1] UTCTime OPTIONAL,
            random1                 [2] BIT STRING OPTIONAL,
            random2                 [3] BIT STRING OPTIONAL,
            protected               [4] OCTET STRING }

    StrongCredentials ::= SEQUENCE {
            certification-path      [0] AF.CertificationPath OPTIONAL,
            bind-token              [1] DAS.Token }

    OtherCredentials ::= SEQUENCE {
            authMechanism           [0] LDAPOID,
            authToken               [1] OCTET STRING }

    Controls ::= SEQUENCE OF SEQUENCE {
            controlType             LDAPString,
            criticality             BOOLEAN DEFAULT FALSE,
            controlValue            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 3 of the LDAP protocol. Note that there is no version negotiation, and the client should just set this parameter to the version it desires. The client may request version 2, in which case the server should implement only the protocol as described in [2].

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

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

- serviceControls: additional requests the client may make about the protocol.

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.

Unlike LDAP v2, the client need not send a Bind Request in the first PDU of the connection. The client may request any operations and the server should treat these as unauthenticated (or authentication may have already occured at a lower layer). If the server requires that the client bind first, the server should reject any request other that binding or unbinding with the "operationsError" result. If the client did not bind before sending a request and receives an operationsError, it should close the connection, reopen it and begin again by first sending a PDU with a Bind Request. This will aid in interoperating with LDAPv2 servers.

Clients may send multiple bind requests on an association. Authentication or controls from earlier binds should subequently be ignored.

4.2.1 Authentication

If no authentication is to be performed, or has been performed at a lower layer, then the simple authentication option should be chosen, and the password be of zero length.

The "simple" authentication option provides minimal authentication facilities, with the contents of the authentication field consisting only of a cleartext password.

The ProtectedPassword authentication option allows a hash of the password, combined optionally with the current time and a random number, to be sent to the DSA. The protected field contains the hash value.

Strong authentication to the directory can be accomplished using the strong credentials option. The ASN.1 type "CertificationPath" is defined in [12], and the ASN.1 type "Token" is defined in [13]. They are included in Appendix B for reference.

Other kinds of authentication to the directory can be performed using the other credentials option. The authMechanism must be the dotted-decimal printable representation of an OBJECT IDENTIFIER of that authentication mechanism: for interoperability the full decimal format must be used. The authToken is arbitrary information of a form defined by that authentication mechanism, encoded in an OCTET STRING.

4.2.1.1. Strong Credentials Signature Algorithm

It is recommended for interoperability that if strong authentication is to be performed, then if the server's or client's certificates contain RSA public keys the PKCS md5WithRSAEncryption (1.2.840.113549.1.1.4) algorithm should be used.

4.2.2. Service Controls

Service Controls are requests made by the client which affect its interaction with the server. Controls are not saved after a session unbinds or disconnects abruptly, and do not affect other sessions to this or other servers.

If the server is not capable of setting one or more requested controls, it should set as many as possible. If any of the controls which the server could not set are marked as critical, it should return the unavailableCriticalExtension error.

The controlType field must either be a string defined in this section, or a dotted-decimal representation of an OBJECT IDENTIFIER. This will aid in preventing conflicts between privately-defined control extensions.

The following controls have been defined:

- referringServer
- chainingProhibited
- supportedReferral
- useAliasOnUpdate
- manageDsaIT
- preferredLanguage

The referringServer control is always non-critical. The value field contains the URL of another server which referred an operation to this server. This control should only be present if the connection is being made only to process a referral. If the connection will be held open to handle referrals from multiple servers this control should be omitted.

The chainingProhibited control may be critical or non-critical at the clients request. The value should be an empty string. If present, the server should not contact any other servers, if it would be possible to instead return to the client a referral. If the server is a gateway to X.500, it should set the chainingProhibited service control on any DAP/DSP requests it makes.

The supportedReferral control is always non-critical. The field is a string name of a protocol which the client implements. The name of the protocol may be "ldap", "cldap", "dap", or any IANA-assigned protocol name or URL mechanism. If this control is present, a server should return a referral rather than chain to another server.

The useAliasOnUpdate control may be critical or non-critical at the clients request. The value should be an empty string. If present, the server should permit alias names to be used as components of a

Distinguished Name in Add, Modify and Delete operations. If the server is a gateway to X.500, it should set the useAliasOnUpdate critical extensions on any DAP/DSP AddEntry, ModifyEntry and RemoveEntry requests it makes.

The manageDsaIT control is always critical. The value should be an empty string. If present, the chainingProhibited control must also be present and critical. This control affects the name resolution behavior of the server to permit a manager to read and modify knowledge references and other server-specific attributes. If the server is a gateway to X.500, it should set the manageDsaIT critical extension, as well as the appropriate common arguments, on any DAP/DSP requests it makes.

The preferredLanguage control is always non-critical. The value is an ISO 646 language code (such as "EN" for English). This control advises the server what language should be used for returned attribute values and error messages. It does not affect character sets; BMP is always used.

4.2.3. Bind Response

The Bind Response will be one of the following, either BindResponseBasic or BindResponseExtended.

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

A BindResponseBasic consists simply of an indication from the server of the status of the client's request for the initiation of a protocol session. If the bind was successful, the resultCode will be success, otherwise it will be one of:

```
operationsError
protocolError
authMethodNotSupported
strongAuthRequired
referral
inappropriateAuthentication
invalidCredentials
unavailable
unavailableCriticalExtension
```

If the client receives a BindResponseBasic response where the resultCode was not success, it should close the connection as the server will be unwilling to accept further operations.

The BindResponseExtended is used to provide additional information in the bind response, for either a successful or unsuccessfull bind operation.

```
BindResponseExtended ::= [APPLICATION 17] SEQUENCE {
        result          [0] LDAPResult,
```

```
        serverURL         [1] LDAPURL OPTIONAL,
        serverCreds       [2] AuthenticationChoice OPTIONAL,
        supportedExtns    [3] SEQUENCE OF LDAPString,
        unsupportedCtls   [4] SEQUENCE OF LDAPString }
```

The serverURL contains the URL of this LDAP server. The serverCreds allows the client to authenticate the server to which it is communicating. The supportedExtns contains the names of service controls and extended operations which this server supports. The unsupportedCtls names the service controls which the client requested but the server was not able to set.

4.2.4 Bind Continuation

The BindContinuationRequest is used when a "challenge-response" style of authentication is to be performed. The client will initially send a BindRequest, and will receive a BindResponseExtended. The client may then send a BindContinuationRequest to supply additional information as part of a single authentication process. The server will reply to the BindContinuationRequest with a BindResponseExtended.

```
    BindContinuationRequest ::= [APPLICATION 19] SEQUENCE {
            otherkind       [6] OtherCredentials }
```

4.3. Unbind Operation

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

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

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

4.4. 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),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList,
        pageSizeLimit   [0] INTEGER OPTIONAL,
        sortKeys        [1] SortKeyList OPTIONAL,
        modifyRightsReq [3] BOOLEAN DEFAULT FALSE,
        extraAttributes [4] BOOLEAN DEFAULT FALSE,
        attrSizeLimit   [5] INTEGER OPTIONAL,
        subentries      [6] BOOLEAN DEFAULT FALSE,
        dontUseCopy     [7] BOOLEAN DEFAULT FALSE,
        usePartialCopy  [8] BOOLEAN DEFAULT FALSE }

SortKeyList ::= SEQUENCE OF SEQUENCE {
        attributeType   AttributeType,
        orderingRule    [0] MatchingRuleId OPTIONAL,
        reverseOrder    [1] BOOLEAN DEFAULT FALSE }
```



```
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,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type            AttributeType,
        substrings      SEQUENCE SIZE (1..maxInt) OF CHOICE {
                initial [0] LDAPString,
                any     [1] LDAPString,
                final   [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRules   [1] SET SIZE (0..maxInt) OF MatchingRuleId,
        type            [2] AttributeType,
        matchValue      [3] AssertionValue,
```

dnAttributes [4] BOOLEAN }

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:

  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.

- typesOnly: 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. The and, or and not choices may be used to form boolean combinations of filters.

- 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.

- pageSizeLimit: if present and set to TRUE, then if more entries are to be returned than the pageSizeLimit, the server should return only as many as this limit before returning the SearchResultDone response. It must cache all of the results for the lifetime of the association. (The client will be able to request more of the entries using the ResumeRequest, and the cached results can be cleared if the client sends the Abandon operation for this search). If the same or fewer entries than this limit are to be returned, the server should return all the entries and the SearchResultDone response, and need not cache the result. The pageSizeLimit does not affect SearchResultReference responses, of which any number may be returned by the server.

- sortKeys: If this field is present, then it specifies one or more attribute types and matching rules, and the returned entries should be sorted in order based on these types. If the reverseOrder field is set to TRUE, then the entries will be presented in reverse sorted order.

  If the server does not recognize any of the attribute types, or the ordering rule associated with an attribute type is not applicable, or none of the attributes in the search responses are of these types, then the sortKeys field is ignored and result entries are returned in random order.

- modifyRightsReq: If this field is set to TRUE and the scope field is set to baseObject, then the client requests that the modification rights for the entry be included in the search result. Support for this field is optional, and clients should expect that not all servers will implement returning modify rights.

- extraAttributes: If this field is present and set to TRUE then all operational attributes are requested to be returned as well. Note that specific operational attributes may instead be listed in the attributes field. Servers are permitted to ignore extraAttributes if returning this information is prohibited by security policy. Clients should note that many operational attributes are not modifiable.

- attrSizeLimit: If this field is present, then if the size in bytes of an attribute and all its values which would be returned in a result entry exceeds this size in bytes, then the attribute is not included in the result and the incompleteEntry field is set to TRUE.

- subentries: if present and set to TRUE, the server should ignore ordinary entries and only perform the search against subentries. If

the server not support subentries and this field is TRUE it should not do any searching, and return an empty result. (Note that if the LDAP is backed by an X.500(1988) directory service, the LDAP server may receive a protocolError or unavailableCriticalExtension error, which it should discard and instead return to the client an empty result.)

- dontUseCopy: if present and set to TRUE, only the server which holds the master copy of the entry is permitted to perform the filtering and attribute selection.

- usePartialCopy: if present and set to TRUE, if the server holds a shadow copy of at least one attribute from a matching entry, it should use that copy to satisfy the search, even if not all the attributes requested are present in the shadowed copy.

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

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
        attributes      PartialAttributeList,
        modifyRights    [2] BOOLEAN OPTIONAL,
        incompleteEntry [3] BOOLEAN DEFAULT FALSE,
        fromEntry       [4] BOOLEAN DEFAULT FALSE,
        thisEntryNumber [5] INTEGER OPTIONAL,
        totalCount      [6] INTEGER OPTIONAL }

PartialAttributeList ::= SEQUENCE SIZE (0..maxInt) OF SEQUENCE {
        type    AttributeDescription,
        vals    SET SIZE (0..maxInt) OF AttributeValue }

SearchResultReference ::= [APPLICATION 18] Referral

SearchResultDone ::= [APPLICATION 5] LDAPResult

SearchResultFull ::= SEQUENCE SIZE (1..maxInt) OF CHOICE {
                entry           SearchResultEntry,
                reference       SearchResultReference,
                resultCode      SearchResultDone }
```

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

If the LDAP session is operating over a connection-oriented transport such as TCP, the server will return to the client a sequence of responses in separate LDAP messages. There may be zero or more responses containing SearchResultEntry, one for each entry found during the search. There may also be zero or more responses containing SearchResultReference, one for each area not explored by this server during the search. The SearchResultEntry and

SearchResultReferences may come in any order. Following all the SearchResultReference responses and all SearchResultEntry responses up to a pageSizeLimit (if any), the server will return a response containing the SearchResultDone, which contains an indication of success, or detailing any errors that have occurred.

If the LDAP session is operating over a connectionless transport such as UDP, the server will return to the client only one response, a LDAPMessage containing a SearchResultFull data type. All if any but the last element of the SEQUENCE OF must be of the SearchResultEntry type, and the last must be of the SearchResultDone type.

The SearchResultFull is never returned over a connection-oriented transport.

Each entry returned in a SearchResultEntry will contain all attributes, complete with associated values if necessary, as specified in the attributes field of the Search Request. Return of attributes is subject to access control and other administrative policy.

In a SearchResultEntry, as an encoding optimisation, the value of the objectName LDAP DN may use a trailing '*' character to refer to the baseObject of the corresponding searchRequest. For example, if the baseObject is specified as "o=UofM, c=US", then the following objectName LDAPDNs in a response would have the indicated meanings

objectName returned	actual LDAPDN denoted
"*"	"o=UofM, c=US"
"cn=Babs Jensen, *"	"cn=Babs Jensen, o=UofM, c=US"

If (and only if) the modifyRightsReq field was present in the Search Request may the server also include the ModifyRights field in the entry. If present and set to TRUE, then the server suggests it is likely that a valid modification operation on this entry would succeed. If present and set to FALSE, then it is likely the operation would fail due to an authentication or access control restriction. If no information is available the server should not include the modifyRights field in the response.

The incompleteEntry flag is set if one or more attributes are not present in the PartialAttributeList, because their size would have exceeded the attribute size limit, or if a partial shadow copy of the entry was used to satisfy the request and some requested attributes are not returned. It is never set just because typesOnly was set to TRUE.

The server may set the fromEntry field in a SearchResult entry to TRUE if it is known that the search is not based upon a shadow or cached

copy of the entry, but that the source of entry data has been directly contacted.

If the pageSizeLimit control was present, the server must number the entries which match the search. The first entry returned will have thisEntryNumber field contain the number 0, the next is number 1, etc. The server must also indicate a count of the total number of entries in the field totalCount. The server may revise the count, a larger totalCount field in a later SearchResultEntry will override the totalCount field of an earlier SearchResultEntry for that search.

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

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.

4.5. Resume Search Operation

The Resume Search Operation is used in conjunction with a Search operation which was previously issued on this association.

```
ResumeRequest ::= [APPLICATION 20] SEQUENCE {
        searchRequestID         [0] MessageID,
        startAtEntry            [1] INTEGER OPTIONAL,
        entriesToReturn         [2] INTEGER OPTIONAL }
```

The SearchRequest must have been made with the pageSizeLimit field present, and the server must not have returned the SearchResultDone for this search, indicating that all the results have been returned or an error was encountered. A Search which has been abandoned cannot be resumed.

The searchRequestID field must contain the value of messageID which the client used for the original search operation.

The startAtEntry number may be any number greater than 0, and the sum of startAtEntry and entriesToReturn must not be greater than the value of totalCount returned by the server for this search. Note that the client may request that the server retransmit entries which it has already sent, by setting a value of startAtEntry smaller than the thisEntryNumber of

the last entry which the server has transmitted.

The server will respond to the ResumeRequest with either a ResumeError, or with a series of SearchResultEntry responses. The ResumeError is only returned if the server detected a problem with the ResumeRequest, such as an invalid searchRequestID. The SearchResultEntry responses have the MessageID of the ResumeRequest, not of the original SearchRequest.

```
ResumeError ::= [APPLICATION 21] LDAPResult
```

An example of using Resume is as follows:

```
          CLIENT                              SERVER


          0,BindRequest
                              -->
                                              0,BindResponse
                              <--

          1,SearchRequest (pageSizeLimit=2)
                              -->
                                              (search matches 5 entries)
                                              1,SearchResultEntry (0 of 5)
                              <--
                                              1,SearchResultEntry (1 of 5)
                              <--
                                              1,SearchResultDone
                              <--
```



```
          2,ResumeRequest (search id 1, start at 2, retrieve 3)
                              -->
                                              2,SearchResultEntry (2 of 5)
                              <--
                                              2,SearchResultEntry (3 of 5)
                              <--
                                              2,SearchResultEntry (4 of 5)
                              <--

          3,AbandonRequest (id 1)
                              -->
                                              (search cache cleared)
```

4.6. 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 SIZE (1..maxInt) OF SEQUENCE {
                operation       ENUMERATED {
                                        add     (0),
                                        delete  (1),
                                        replace (2) },
                modification    AttributeTypeAndValues } }

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

Parameters of the Modify Request are:

- object: The object to be modified. The value of this field should name the object to be modified. 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. A replace with no value should delete the entire attribute.

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.

4.7. 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,
        targetSystem    [0] LDAPString OPTIONAL }

AttributeList ::= SEQUENCE SIZE (1..maxInt) OF SEQUENCE {
        type    AttributeDescription,
        vals    SET SIZE (1..maxInt) 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. Note also that the server will not dereference any aliases in locating the entry to be added, and that there are never any entries subordinate to an alias entry.

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

- targetSystem: if present, the string representation of an AccessPoint93, identifying the server which should hold the target entry. If the server does not support the targetSystem extension it should return the error unavailableCriticalExtension.

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.

4.8. Delete Operation

The Delete Operation allows a client to request the removal of an

entry from the directory. The Delete Request is defined as follows:

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


The Delete Request consists of the Distinguished Name of the entry to be deleted. Note that the server will not dereference aliases while resolving the name of the target entry to be removed.

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.

4.9. Modify DN Operation

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

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

Parameters of the Modify DN Request are:

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

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

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

- newSuperior: if present, this is the name of another entry which should be the superior of the subtree in the entry field.

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

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

Upon receipt of a 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 DN Response. The attributes that make up the old RDN are deleted from the entry, or kept, depending on the setting of the deleteoldrdn parameter.

4.10. Compare Operation

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

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion,
        dontUseCopy     [1] BOOLEAN DEFAULT FALSE }
```


Parameters of the Compare Request are:

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

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

- dontUseCopy: if present and set to TRUE, only the server which holds the master copy of the entry is permitted to return the compareTrue or compareFalse results.

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

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

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

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

4.11. Abandon Operation

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

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

The MessageID must be that of a Search, Resume or Compare operation which was requested earlier during this association. Other types of operations cannot be abandoned.

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

If the MessageID is for a Search operation in which pageSizeLimit was set, the abandon will clear the results from the server's cache. Abandoning a Resume operation does not clear the cache.

4.11 Extended Operation

It may be desirable in some communities to define additional operations for services not available in this protocol, for instance digitally signed operations and results. Thus an extension mechanism

The extended operation allows clients to make requests and receive responses with bilaterally-defined syntaxes and semantics.


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

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

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

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        responseName            [0] LDAPOID OPTIONAL,
        response                [1] OCTET STRING OPTIONAL,
        standardResponse        [2] LDAPResult }
```

If the server does not recognize the operation name, it should return only the standardResponse field, containing the protocolError result code.

5. Protocol Element Encodings and Transfer

For compatibility with the existing LDAP v2 and CLDAP protocols, four underlying services are defined here. However an LDAP server need not implement all of them.

5.1. Mapping Onto BER-based 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.

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

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

(2) BIT STRINGs and OCTET STRINGs will be encoded in the primitive form only.

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

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

If an implementation supports the protected or strong authentication elements then the following additional restrictions apply:

(5) UTC Times in the protocol itself should be encoded with the "Z" suffix, not as a local time. (This requirement does not apply to times in attribute values).

(6) Unused bits in the final octet of the encoding of a BIT STRING value, if there are any, should always be set to zero.

These restrictions do not apply to ASN.1 types encapsulated inside of OCTET STRINGs, such as attribute values.

5.1.1. 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.

5.1.2. Connection Oriented Transport Service (COTS)

The connection is established. No special use of T-Connect is made.

Each LDAPMessage PDU is mapped directly onto T-Data.

5.1.3. User Datagram Protocol (UDP)

The LDAPMessage PDUs are mapped directly onto UDP datagrams. Only one request may be sent in a single datagram. Only one response may be sent in a single datagram. Server implementations running over the UDP should provide a protocol listener on port 389.

The only operations which the client may request are searchRequest and abandonRequest. The server may only respond with the searchResultFull.

5.1.4. Secure Socket Layer over TCP (SSL)

After establishing the SSL connection over TCP, the LDAPMessage PDUs are mapped directly onto the bytestream. Server implementations running over SSL/TCP should provide a protocol listener on port TBD.

6. Implementation Guidelines

6.1. Server Implementations

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

6.2. Client Implementations

For simple lookup applications using the connectionless transport protocol UDP, use of a retry algorithm with multiple servers similar to that commonly used in DNS stub resolver implementations is recommended. The location of a CLDAP server or servers may be better specified using IP addresses (simple or broadcast) rather than names that must first be looked up in another directory such as DNS.

7. Security Considerations

When used with a connection-oriented transport, this version of the protocol provides facilities for the LDAP v2 authentication mechanism: simple authentication using a cleartext password. It also provides for two other authentication mechanisms as described in X.511: transfer of a hash of the client's password, and strong authentication based on the private key of the client. It is also permitted that the server can return its credentials to the client.

This document also defines a mapping of LDAP over the Secure Sockets Layer (SSL), which can provide strong authentication, integrity and privacy of the connection.

Use of cleartext password is strongly discouraged where the underlying transport service cannot guarantee confidentiality.

When used with the connectionless transport, no security services are available. There has been some discussion about the desirability of authentication with connectionless LDAP requests. This might take the form of a clear text password (which would go against the current IAB drive to remove such things from protocols) or some arbitrary credentials. It is felt that, in general, authentication would incur sufficient overhead to negate the advantages of the connectionless basis of CLDAP. If an application requires authenticated access to the directory then CLDAP is not an appropriate protocol.

8. Acknowledgements

This document is an update to RFC 1777, by Wengyik Yeong, Tim Howes, and Steve Kille. It also includes material from RFC 1798, by Alan Young. Design ideas included in this document are based on those discussed in ASID and other IETF Working Groups.

9. Bibliography

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

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

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

[4] S. Kille, "A String Representation of Distinguished Names", RFC 1779, March 1995.

[5] M. Wahl, A. Coulbeck, T. Howes, S. Kille, W. Yeong, C. Robbins, "Lightweight X.500 Directory Access Protocol Standard and Pilot Attribute Definitions", <draft-ietf-asid-ldapv3-attributes-03.txt>, May 1996.

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

[7] The Directory: Selected Attribute Types. ITU-T Recommendation X.520, 1993.

[9] T. Howes, M. Smith, An LDAP URL Format, December 1995, <draft-ietf-asid-ldap-format-03.txt>

[10] The Directory: Procedures for Distributed Operation. ITU-T Recommendation X.518, 1993.

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

[12] The Directory: Authentication Framework. ITU-T Recommendation X.509, 1993.

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

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

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

[16] M. Davis, UTF-8, (WG2 N1036) DAM for ISO/IEC 10646-1.

10. Authors' Address

Mark Wahl
ISODE Consortium Inc.
3925 West Braker Lane, Suite 333
Austin, TX 78759
USA

Phone: +1 512-305-0280
EMail: M.Wahl@isode.com

Tim Howes
Netscape Communications Corp.
685 Middlefield
Mountain View, CA 94043
USA

Phone: +1 415 254-1900
EMail: howes@netscape.com

Steve Kille
ISODE Consortium
The Dome, The Square
Richmond
TW9 1DT
UK

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

Appendix A - Complete ASN.1 Definition

In the IMPORTS statement the "AF" module refers to X.509(1993), and the "DAS" module to X.511(1993).

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

        BEGIN

        IMPORTS CertificationPath FROM AF
                Token FROM DAS;

                --- to be provided ---

        END
```

Appendix B - Imported ASN.1 Definitions

Note that the types described here are distinct from those defined in the body of this document.

B.1. Types from X.509(1993) "Authentication Framework"

The type "Certificate" is defined in X.509(1993). It it strongly recommended that clients and server implementations which support certificates implement the draft addendums to X.509 which provide certificate extensions.


```
        AlgorithmIdentifier ::= SEQUENCE {
                algorithm  OBJECT IDENTIFIER,
                parameters ANY OPTIONAL }

        CertificatePair ::= SEQUENCE {
                forward [0] Certificate OPTIONAL,
                reverse [1] Certificate OPTIONAL
                        -- at least one of the pair shall be present -- }

        CertificationPath ::= SEQUENCE {
                userCertificate         Certificate,
                theCACertificates       SEQUENCE OF CertificatePair
                                                OPTIONAL }
```

B.2. Types from X.511(1993) "Directory Abstract Syntax"

The type "DistinguishedName" is defined in X.501(1993). It is the ASN.1 encoding, not a string encoding.

```
        Token ::= SIGNED { SEQUENCE {
                algorithm       [0] AlgorithmIdentifier,
                name            [1] DistinguishedName,
                time            [2] UTCTime,
                random          [3] BIT STRING } }
```