LDAP: Directory Information Models
<draft-ietf-ldapbis-models-05.txt>

Status of this Memo

Abstract

   The Lightweight Directory Access Protocol (LDAP) is an Internet
   protocol for accessing distributed directory services which act in
   accordance with X.500 data and service models. This document
   describes the X.500 Directory Information Models, as used in LDAP.

Table of Contents

1. Introduction

This document discusses the X.500 Directory Information Models
[X.501], as used by the Lightweight Directory Access Protocol (LDAP)
[Roadmap].

The Directory is "a collection of open systems cooperating to provide
directory services" [X.500]. The information held in the Directory is
collectively known as the Directory Information Base (DIB). A
Directory user, which may be a human or other entity, accesses the
Directory through a client (or Directory User Agent (DUA)). The
client, on behalf of the directory user, interacts with one or more
servers (or Directory System Agents (DSA)). A server holds a fragment
of the DIB.

The DIB contains two classes of information:

   1) user information (e.g., information provided and administrated
      by users). Section 2 describes the Model of User Information.

   2) administrative and operational information (e.g., information
      used to administer and/or operate the directory). Section 3
      describes the model of Directory Administrative and Operational
      Information.

These two models, referred to as the generic Directory Information
Models, describe how information is represented in the Directory.
These generic models provide a framework for other information models.
Section 4 discusses the subschema information model and subschema
discovery. Section 5 discusses the DSA (Server) Informational Model.

Other X.500 information models, such as access control, collective
attribute, distribution knowledge, and replication knowledge
information models, may be adapted for use in LDAP. Specification of
how these models are to be used in LDAP is left to future documents.

1.1. Relationship to Other LDAP Specifications

This document is a integral part of the LDAP technical specification
[Roadmap] which obsoletes the previously defined LDAP technical
specification, RFC 3377, in its entirety.

This document obsoletes RFC 2251 sections 3.2 and 3.4, as well as
portions of sections 4 and 6. Appendix A.1 summaries changes to these
sections. The remainder of RFC 2251 is obsoleted by the [Protocol],
[AuthMeth], and [Roadmap] documents.

This document obsoletes RFC 2252 sections 4, 5 and 7. Appendix A.2
summaries changes to these sections. The remainder of RFC 2252 is
obsoleted by [Syntaxes] and [Schema].

This document obsoletes RFC 2256 sections 5.1, 5.2, 7.1 and 7.2.
Appendix A.3 summarizes changes to these sections. The remainder of
RFC 2256 is obsoleted by [Schema] and [Syntaxes].

1.2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in BCP 14 [RFC2119].

Schema definitions are provided using LDAP description formats (as
defined in Section 4.1). Definitions provided here are formatted
(line wrapped) for readability. Matching rules and LDAP syntaxes
referenced in these definitions are specified in [Syntaxes].

1.3. Common ABNF Productions

A number of syntaxes in this document are described using Augmented
Backus-Naur Form (ABNF) [RFC2234]. These syntaxes (as well as a
number of syntaxes defined in other documents) rely on the following
common productions:

```
    keystring = leadkeychar *keychar
    leadkeychar = ALPHA
    keychar = ALPHA / DIGIT / HYPHEN

    number = DIGIT / ( LDIGIT 1*DIGIT )

    ALPHA  = %x41-5A / %x61-7A   ; "A"-"Z" / "a"-"z"
    DIGIT  = %x30 / LDIGIT       ; "0"-"9"
    LDIGIT = %x31-39             ; "1"-"9"

    HEX     = DIGIT / %x41-46 / %x61-66 ; 0-9 / A-F / a-f

    SP     = 1*SPACE  ; one or more " "
    WSP    = 0*SPACE  ; zero or more " "

    NULL   = %x00 ; null (0)
    SPACE  = %x20 ; space (" ")
    DQUOTE = %x22 ; quote (""")
    SHARP  = %x23 ; octothorpe (or sharp sign) ("#")
    DOLLAR = %x24 ; dollar sign ("$")
```

```
      SQUOTE = %x27 ; single quote ("'")
      LPAREN = %x28 ; left paren ("(")
      RPAREN = %x29 ; right paren (")")
      PLUS   = %x2B ; plus sign ("+")
      COMMA  = %x2C ; comma (",")
      HYPHEN = %x2D ; hyphen ("-")
      DOT    = %x2E ; period (".")
      SEMI   = %x3B ; semicolon (";")
      LANGLE = %x3C ; left angle bracket ("<")
      EQUALS = %x3D ; equals sign ("=")
      RANGLE = %x3E ; right angle bracket (">")
      X      = %x58 ; uppercase x ("X")
      ESC    = %x5C ; backslash ("\")
      USCORE = %x5F ; underscore ("_")
      LCURLY = %x7B ; left curly brace "{"
      RCURLY = %x7D ; right curly brace "}"

      ; Any UTF-8 character
      UTF8   = UTF1 / UTFMB
      UTFMB  = UTF2 / UTF3 / UTF4 / UTF5 / UTF6
      UTF0   = %x80-BF
      UTF1   = %x00-7F
      UTF2   = %xC0-DF 1(UTF0)
      UTF3   = %xE0-EF 2(UTF0)
      UTF4   = %xF0-F7 3(UTF0)
      UTF5   = %xF8-FB 4(UTF0)
      UTF6   = %xFC-FD 5(UTF0)

      ; Any octet
      OCTET  = %x00-FF
```

Object identifiers are represented in LDAP using a dot-decimal format
conforming to the ABNF:

```
      numericoid = number *( DOT number )
```

Short names, also known as descriptors, are used as more readable
aliases for object identifiers. Short names are case insensitive and
conform to the ABNF:

```
      descr = keystring
```

Where either an object identifier or a short name may be specified,
the following production is used:

```
      oid = descr / numericoid
```

The <descr> form is preferred. When a production <oid> is encoded (in

a protocol field, attribute value, assertion value, or elsewhere), the
<descr> encoding option SHOULD be used instead of the <numericoid>
encoding option.

2. **Model of Directory User Information**

  As [X.501] states:

      The purpose of the Directory is to hold, and provide access to,
      information about objects of interest (objects) in some 'world'.
      An object can be anything which is identifiable (can be named).

      An object class is an identified family of objects, or conceivable
      objects, which share certain characteristics. Every object belongs
      to at least one class. An object class may be a subclass of other
      object classes, in which case the members of the former class, the
      subclass, are also considered to be members of the latter classes,
      the superclasses. There may be subclasses of subclasses, etc., to
      an arbitrary depth.

  A directory entry, a named collection of information, is the basic
  unit of information held in the Directory. There are multiple kinds
  of directory entries.

  An object entry represents a particular object. An alias entry
  provides alternative naming. A subentry holds administrative and/or
  operational information.

  The set of entries representing the DIB are organized hierarchically
  in a tree structure known as the Directory Information Tree (DIT).

  Section 2.1 describes the Directory Information Tree
  Section 2.2 discusses naming of entries.
  Section 2.3 discusses the structure of entries.
  Section 2.4 discusses object classes.
  Section 2.5 discusses attribute descriptions.
  Section 2.6 discusses alias entries.

2.1. **The Directory Information Tree**

  As noted above, the DIB is composed of a set of entries organized
  hierarchically in a tree structure known as the Directory Information
  Tree (DIT). Specifically, a tree where vertices are the entries.

  The arcs between vertices define relations between entries. If an arc
  exists from X to Y, then the entry at X is the immediate superior of Y

and Y is the immediate subordinate of X. An entry's superiors are the
entry's immediate superior and its superiors. An entry's subordinates
are all of its immediate subordinates and their subordinates.

Similarly, the superior/subordinate relationship between object
entries can be used to derive a relation between the objects they
represent. DIT structure rules can be used to govern relationships
between objects.

Note: An entry's immediate superior is also known as the entry's
      parent and an entry's immediate subordinate is also known as the
      entry's child.

2.2. Naming of Entries

2.2.1. Relative Distinguished Names

Each entry is named relative to its immediate superior. This relative
name, known as its Relative Distinguished Name (RDN) [X.501], is
composed of an unordered set of one or more attribute value assertions
(AVA) consisting of an attribute description with zero options and an
attribute value. These AVAs are chosen from the attributes of the
entry.

An entry's relative distinguished name must be unique among all
immediate subordinates of the entry's immediate superior (i.e., all
siblings).

The following are example string representations of RDNs [LDAPDN]:

    UID=12345
    OU=Engineering
    CN=Kurt Zeilenga+L=Redwood Shores

The last is an example of a multi-valued RDN.

2.2.2. Distinguished Names

An entry's fully qualified name, known as its Distinguished Name (DN)
[X.501], is the concatenation of its RDN and its immediate superior's
DN. A Distinguished Name unambiguously refers to an entry in the
tree. The following are example string representations of DNs
[LDAPDN]:

    UID=nobody@example.com,DC=example,DC=com
    CN=John Smith,OU=Sales,O=ACME Limited,L=Moab,ST=Utah,C=US

2.2.3. **Alias Names**

  An alias, or alias name, is "an name for an object, provided by the
  use of alias entries" [X.501]. Alias entries are described in Section
  2.6.

2.3. **Structure of an Entry**

  An entry consists of a set of attributes which hold information about
  the object which entry represents. Some attributes represent user
  information and are called user attributes. Other attributes
  represent operational and/or administrative information and are called
  operational attributes.

  An attribute is an attribute description (a type and zero or more
  options) with one or more associated values. An attribute is often
  referred to by its attribute description. For example, the
  'givenName' attribute is the attribute which consists of the attribute
  description 'givenName' (the 'givenName' attribute type [Schema] and
  zero options) and one or more associated values.

  The attribute type governs whether the attribute can have multiple
  values, the syntax and matching rules used to construct and compare
  values of that attribute, and other functions. Options indicate
  subtypes and other functions. No two values of an attribute may be
  equivalent.

  Two values are considered equivalent if they would match according to
  the equality matching rule of the attribute type. If the attribute
  type is defined with no equality matching rule, two values are
  equivalent if and only if they are identical.

  For example, the 'givenName' attribute can have can have more than one
  value, they must be Directory Strings, and they are case insensitive.
  The 'givenName' attribute cannot hold both "John" and "JOHN" as these
  are equivalent values per the equality matching rule of the attribute
  type.

2.4. **Object Classes**

  An object class is "an identified family of objects (or conceivable
  objects) which share certain characteristics" [X.501].

  As defined in [X.501]:

     Object classes are used in the Directory for a number of purposes:

- describing and categorising objects and the entries that
  correspond to these objects;

- where appropriate, controlling the operation of the Directory;

- regulating, in conjunction with DIT structure rule
  specifications, the position of entries in the DIT;

- regulating, in conjunction with DIT content rule
  specifications, the attributes that are contained in entries;

- identifying classes of entry that are to be associated with a
  particular policy by the appropriate administrative authority.

An object class (a subclass) may be derived from an object class
(its direct superclass) which is itself derived from an even more
generic object class. For structural object classes, this process
stops at the most generic object class, 'top' (defined in Section
2.4.1). An ordered set of superclasses up to the most superior
object class of an object class is its superclass chain.

An object class may be derived from two or more direct
superclasses (superclasses not part of the same superclass chain).
This feature of subclassing is termed multiple inheritance.

Each object class identifies the set of attributes required to be
present in entries belonging to the class and the set of attributes
allowed to be present in entries belonging to the class. As an entry
of a class must meet the requirements of each class it belongs to, it
can be said that an object class inherits the sets of allowed and
required attributes from its superclasses. A subclass can identify an
attribute allowed by a subclass as being required. If an attribute is
a member of both sets, it is required to be present.

Each object class is defined to be one of three kinds of object
classes: Abstract, Structural, and Auxiliary.

Each object class is identified by an object identifier (OID) and,
optionally, one or more short names (descriptors).

2.4.1. Abstract Object Classes

An abstract object class, as the name implies, provides a base of
characteristics from which other object classes can be defined to
inherit from. An entry cannot belong to an abstract object class
unless it belongs to a structural or auxiliary class which inherits
from that abstract class.

   Abstract object classes can not derive from structural nor auxiliary
   object classes.

   All structural object classes derive (directly or indirectly) from the
   'top' abstract object class. Auxiliary object classes do not
   necessarily derive from 'top'.

        (2.5.6.0 NAME 'top' ABSTRACT MUST objectClass)

   All entries belong to the 'top' abstract object class.

2.4.2. Structural Object Classes

   As stated in [X.501]:

        An object class defined for use in the structural specification of
        the DIT is termed a structural object class. Structural object
        classes are used in the definition of the structure of the names
        of the objects for compliant entries.

        An object or alias entry is characterised by precisely one
        structural object class superclass chain which has a single
        structural object class as the most subordinate object class.
        This structural object class is referred to as the structural
        object class of the entry.

        Structural object classes are related to associated entries:

          - an entry conforming to a structural object class shall
            represent the real-world object constrained by the object
            class;

          - DIT structure rules only refer to structural object classes;
            the structural object class of an entry is used to specify the
            position of the entry in the DIT;

          - the structural object class of an entry is used, along with an
            associated DIT content rule, to control the content of an
            entry.

        The structural object class of an entry shall not be changed.

   Each structural object class is a (direct or indirect) subclass of the
   'top' abstract object class.

   Structural object classes cannot subclass auxiliary object classes.

Each entry is said to belong to its structural object class as well as
all classes in its structural object class's superclass chain, which
always includes 'top'.

2.4.3. Auxiliary Object Classes

Auxiliary object classes are used augment the characteristics of
entries. They are commonly used to augment the sets of attributes
required and allowed attributes to be present in an entry. They can
be used to describe entries or classes of entries.

Auxiliary object classes cannot subclass structural object classes.

Each entry can belong to any number of auxiliary object classes. The
set of auxiliary object classes which an entry belongs to can change
over time.

2.5. Attribute Descriptions

An attribute description is composed of an attribute type (see Section
2.5.1) and a set of zero or more attribute options (see Section
2.5.2).

An attribute description is represented by the ABNF:

        attributedescription = attributetype options

        attributetype = oid

        options = *(SEMI option)

        option = 1*keychar

where <attributetype> identifies the attribute type and each <option>
identifies an attribute option. Both <attributetype> and <option>
productions are case insensitive. The order in which <option>s appear
is irrelevant. That is, any two <attributedescription>s which consist
of the same <attributetype> and same set of <option>s are equivalent.

Examples of valid attribute descriptions:

        2.5.4.0
        cn;lang-de;lang-en
        owner

An attribute description which consisting of an unrecognized attribute

type is to be treated as unrecognized. Servers SHALL treat an
attribute description with an unrecognized attribute option as
unrecognized. Clients MAY treat an unrecognized attribute option as a
tagging option (see Section 2.5.2.1).

All attributes of an entry must have distinct attribute descriptions.

2.5.1. Attribute Types

An attribute type governs whether the attribute can have multiple
values, the syntax and matching rules used to construct and compare
values of that attribute, and other functions.

The attribute type indicates whether the attribute is a user attribute
or an operational attribute. If operational, the attribute type
indicates the operational usage and whether the attribute can
modifiable by users or not. Operational attributes discussed in
Section 3.4.

An attribute type (a subtype) may derive from another attribute type
(a direct supertype).   The subtype inherits the matching rules and
syntax of its supertype. An attribute type cannot be a subtype of an
attribute of different usage.

An attribute description consisting of a subtype and no options is
said to the direct description subtype of the attribute description
consisting of the subtype's direct supertype and no options.

Each attribute type is identified by an object identifier (OID) and,
optionally, one or more short names (descriptors).

2.5.2. Attribute Options

There are multiple kinds of attribute description options. The LDAP
technical specification details one kind: tagging options.

Not all options can be associated with attributes held in the
directory. Tagging options can be.

Not all options can be use in conjunction with all attribute types.
In such cases, the attribute description is to be treated as
unrecognized.

An attribute description that contains mutually exclusive options
shall be treated as unrecognized. That is, "cn;x-bar;x-foo", where
"x-foo" and "x-bar" are mutually exclusive, is to be treated as

unrecognized.

Other kinds of options may be specified in future documents. These
documents must detail how new kinds of options they define relate to
tagging options. In particular, these documents must detail whether
or not new kinds of options can be associated with attributes held in
the directory, how new kinds of options affect transfer of attribute
values, and how new kinds of options are treated in attribute
description hierarchies.

Options are represented as short case insensitive textual strings
conforming to the <option> production defined in Section 2.5 of this
document.

Procedures for registering options are detailed in BCP 64 [RFC3383].

2.5.2.1. Tagging Options

Attributes held in the directory can have attribute descriptions with
one or more tagging options. Tagging options are never mutually
exclusive.

An attribute description with N tagging options is considered a direct
(description) subtype of all attribute descriptions of the same
attribute type and all but one of the N options. If the attribute
type has a supertype, then the attribute description is also
considered a direct (description) subtype of the attribute description
of the supertype and the N tagging options. That is,
'cn;lang-de;lang-en' is considered a direct subtype of 'cn;lang-de',
'cn;lang-en', and 'name;lang-de;lang-en' ('cn' is a subtype of 'name',
both are defined in [Schema]).

2.5.3. Attribute Description Hierarchies

An attribute description can be the direct subtype of zero or more
other attribute descriptions as indicated by attribute type subtyping
(as described in Section 2.5.1) or attribute tagging option subtyping
(as described in Section 2.5.2.1). These subtyping relationships are
used to form hierarchies of attribute descriptions and attributes.

As adapted from [X.501]:

> Attribute hierarchies allow access to the DIB with varying degrees
> of granularity. This is achieved by allowing the value components
> of attributes to be accessed by using either their specific
> attribute description (a direct reference to the attribute) or by

a more generic attribute description (an indirect reference).

Semantically related attributes may be placed in a hierarchical relationship, the more specialized being placed subordinate to the more generalized. Searching for, or retrieving attributes and their values is made easier by quoting the more generalized attribute description; a filter item so specified is evaluated for the more specialized descriptions as well as for the quoted description.

Where subordinate specialized descriptions are selected to be returned as part of a search result these descriptions shall be returned if available. Where the more general descriptions are selected to be returned as part of a search result both the general and the specialized descriptions shall be returned, if available. An attribute value shall always be returned as a value of its own attribute description.

All of the attribute descriptions in an attribute hierarchy are treated as distinct and unrelated descriptions for user modification of entry content.

An attribute value stored in a object or alias entry is of precisely one attribute description. The description is indicated when the value is originally added to the entry.

For the purpose of subschema administration of the entry, a required attribute specification is fulfilled if the entry contains a value of an attribute description belonging to an attribute hierarchy if the attribute type of that description is the same as the required attribute's type. That is, a "MUST name" specification is fulfilled by 'name' or 'name;x-tag-option', but is not fulfilled by 'CN' nor by 'CN;x-tag-option'. Likewise, an entry may contain a value of an attribute description belonging to an attribute hierarchy if the attribute type of that description is either explicitly included in the definition of an object class to which the entry belongs or allowed by the DIT content rule applicable to that entry. That is, 'name' and 'name;x-tag-option' are allowed by "MAY name" (or by "MUST name"), but 'CN' and 'CN;x-tag-option' are not allowed by "MAY name" (nor by "MUST name").

For the purposes of other policy administration, unless stated otherwise in the specification of the particular administrative model, all of the attribute descriptions in an attribute hierarchy are treated as distinct and unrelated descriptions.

2.5.4. Attribute Values

Attribute values conform to the defined syntax of the attribute.

When an attribute is used for naming of the entry, one and only one
value of the attribute is selected to appear in the Relative
Distinguished Name. This value is known as a distinguished value.

Only attributes whose descriptions have no options can be used for
naming.

2.6. Alias Entries

As adapted from [X.501]:

   An alias, or an alias name, for an object is a an alternative name
   for an object or object entry which is provided by the use of
   alias entries.

   Each alias entry contains, within the 'aliasedObjectName'
   attribute (known as the 'aliasedEntryName' attribute in X.500]), a
   name of some object. The distinguished name of the alias entry is
   thus also a name for this object.

       NOTE - The name within the 'aliasedObjectName' is said to be
              pointed to by the alias. It does not have to be the
              distinguished name of any entry.

   The conversion of an alias name to an object name is termed
   (alias) dereferencing and comprises the systematic replacement of
   alias names, where found within a purported name, by the value of
   the corresponding 'aliasedObjectName' attribute. The process may
   require the examination of more than one alias entry.

   Any particular entry in the DIT may have zero or more alias names.
   It therefore follows that several alias entries may point to the
   same entry. An alias entry may point to an entry that is not a
   leaf entry and may point to another alias entry.

   An alias entry shall have no subordinates, so that an alias entry
   is always a leaf entry.

   Every alias entry shall belong to the 'alias' object class.

2.6.1. 'alias' object class

Alias entries belong to the 'alias' object class.

```
     ( 2.5.6.1 NAME 'alias'
       SUP top STRUCTURAL
       MUST aliasedObjectName )
```

2.6.2. 'aliasedObjectName' attribute type

The 'aliasedObjectName' attribute holds the name of the entry an alias
points to. The 'aliasedObjectName' attribute is known as the
'aliasedEntryName' attribute in X.500.

```
     ( 2.5.4.1 NAME 'aliasedObjectName'
       EQUALITY distinguishedNameMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
       SINGLE-VALUE )
```

The 'distinguishedNameMatch' matching rule and the DistinguishedName
(1.3.6.1.4.1.1466.115.121.1.12) syntax is defined in [Syntaxes].

3. Directory Administrative and Operational Information

This section discusses select aspects of the X.500 Directory
Administrative and Operational Information model [X.501]. LDAP
implementations MAY support other aspects of this model.

3.1. Subtrees

As defined in [X.501]:

    A subtree is a collection of object and alias entries situated at
    the vertices of a tree. Subtrees do not contain subentries. The
    prefix sub, in subtree, emphasizes that the base (or root) vertex
    of this tree is usually subordinate to the root of the DIT.

    A subtree begins at some vertex and extends to some identifiable
    lower boundary, possibly extending to leaves. A subtree is always
    defined within a context which implicitly bounds the subtree. For
    example, the vertex and lower boundaries of a subtree defining a
    replicated area are bounded by a naming context. Similarly, the
    scope of a subtree defining a specific administrative area is
    limited to the context of an enclosing autonomous administrative
    area.

3.2. Subentries

A subentry is a "special sort of entry, known by the Directory, used
to hold information associated with a subtree or subtree refinement"
[X.501]. Subentries are used in Directory to hold for administrative
and operational purposes as defined in [X.501]. Their use in LDAP is
not detailed in this technical specification, but may be detailed in
future documents.

The term "(sub)entry" in this specification indicates that servers
implementing X.500(93) models are, in accordance with X.500(93), to
use a subentry and that other servers are to use an object entry
belonging to the appropriate auxiliary class normally used with the
subentry (e.g., 'subschema' for subschema subentries) to mimic the
subentry. This object entry's RDN SHALL be formed from a value of the
'cn' (commonName) attribute [Schema].

3.3. The 'objectClass' attribute

Each entry in the DIT has an 'objectClass' attribute.

        (2.5.4.0 NAME 'objectClass'
          EQUALITY objectIdentifierMatch
          SYNTAX 1.3.6.1.4.1.1466.115.121.1.38)

The 'objectIdentifierMatch' matching rule and OBJECT IDENTIFIER
(1.3.6.1.4.1.1466.115.121.1.38) syntax is defined in [Syntaxes].

The 'objectClass' attribute specifies the object classes of an entry,
which (among other things) is used in conjunction with user and system
schema to determine the permitted attributes of an entry. Values of
this attribute can be modified by clients, but the 'objectClass'
attribute cannot be removed.

Servers which follow X.500(93) models SHALL restrict modifications of
this attribute to prevent the basic structural class of the entry from
being changed. That is, one cannot change a 'person' into a
'country'.

When creating an entry or adding an 'objectClass' value to an entry,
all superclasses of the named classes SHALL be implicitly added as
well if not already present. That is, if the auxiliary class 'x-a' is
a subclass of the class 'x-b', adding 'x-a' to 'objectClass' causes
'x-b' to added (if is not already present).

Servers SHALL restrict modifications of this attribute to to prevent a
superclasses of remaining 'objectClass' values from being deleted.
That is, if the auxiliary class 'x-a' is a subclass of the class 'x-b'
and the 'objectClass' attribute contains 'x-a', an attempt to delete

'x-b' from the 'objectClass' attribute is an error.

3.4. Operational attributes

Some attributes, termed operational attributes, are used or maintained
by servers for administrative and operational purposes. As stated in
[X.501]: "There are three varieties of operational attributes:
Directory operational attributes, DSA-shared operational attributes,
and DSA-specific operational attributes."

A directory operational attribute is used to represent operational
and/or administrative information in the Directory Information Model.
This includes operational attributes maintained by the server (e.g.
'subschemaSubentry') as well as operational attributes which hold
values administrated by the user (e.g. 'matchingRuleUse').

A DSA-shared operational attribute is used to represent information of
the DSA Information Model. Its values, if shared between DSAs
(servers) are identical (except during periods of transient
inconsistency).

A DSA-specific operational attribute is used to represent information
of the DSA Information Model. Its values, if shared between DSAs
(servers), need not be identical.

The DSA Information Model operational attributes are detailed in
[X.501].

Operational attributes are not normally visible. They are not
returned in search results unless explicitly requested by name.

Not all operational attributes are user modifiable.

Entries may contain, among others, the following operational
attributes.

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

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

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

  - modifyTimestamp: the time this entry was last modified.

Servers SHOULD maintain the 'creatorsName', 'createTimestamp',

'modifiersName', and 'modifyTimestamp' for all entries of the DIT.

3.4.1. 'creatorsName'

This attribute appears in entries which were added using the protocol
(e.g., using the Add operation). The value is the distinguished name
of the creator.

```
     ( 2.5.18.3 NAME 'creatorsName'
       EQUALITY distinguishedNameMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
       SINGLE-VALUE NO-USER-MODIFICATION
       USAGE directoryOperation )
```

The 'distinguishedNameMatch' matching rule and the DistinguishedName
(1.3.6.1.4.1.1466.115.121.1.12) syntax are defined in [Syntaxes].

3.4.2. 'createTimestamp'

This attribute appears in entries which were added using the protocol
(e.g., using the Add operation). The value is the time the entry was
added.

```
     ( 2.5.18.1 NAME 'createTimestamp'
       EQUALITY generalizedTimeMatch
       ORDERING generalizedTimeOrderingMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
       SINGLE-VALUE NO-USER-MODIFICATION
       USAGE directoryOperation )
```

The 'generalizedTimeMatch' and 'generalizedTimeOrderingMatch' matching
rules and the GeneralizedTime (1.3.6.1.4.1.1466.115.121.1.24) syntax
are defined in [Syntaxes].

3.4.3. 'modifiersName'

This attribute appears in entries which have been modified using the
protocol (e.g., using Modify operation). The value is the
distinguished name of the last modifier.

```
     ( 2.5.18.4 NAME 'modifiersName'
       EQUALITY distinguishedNameMatch
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
       SINGLE-VALUE NO-USER-MODIFICATION
       USAGE directoryOperation )
```

The 'distinguishedNameMatch' matching rule and the DistinguishedName
(1.3.6.1.4.1.1466.115.121.1.12) syntax are defined in [Syntaxes].

3.4.4. 'modifyTimestamp'

This attribute appears in entries which have been modified using the
protocol (e.g., using the Modify operation). The value is the time
the entry was last modified.

```
( 2.5.18.2 NAME 'modifyTimestamp'
  EQUALITY generalizedTimeMatch
  ORDERING generalizedTimeOrderingMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24
  SINGLE-VALUE NO-USER-MODIFICATION
  USAGE directoryOperation )
```

The 'generalizedTimeMatch' and 'generalizedTimeOrderingMatch' matching
rules and the GeneralizedTime (1.3.6.1.4.1.1466.115.121.1.24) syntax
are defined in [Syntaxes].

4. Directory Schema

As defined in [X.501]:

    The Directory Schema is a set of definitions and constraints
    concerning the structure of the DIT, the possible ways entries are
    named, the information that can be held in an entry, the
    attributes used to represent that information and their
    organization into hierarchies to facilitate search and retrieval
    of the information and the ways in which values of attributes may
    be matched in attribute value and matching rule assertions.

    NOTE 1 - The schema enables the Directory system to, for example:

    - prevent the creation of subordinate entries of the wrong
      object-class (e.g. a country as a subordinate of a person);

    - prevent the addition of attribute-types to an entry
      inappropriate to the object-class (e.g. a serial number to a
      person's entry);

    - prevent the addition of an attribute value of a syntax not
      matching that defined for the attribute-type (e.g. a printable
      string to a bit string).

    Formally, the Directory Schema comprises a set of:

   a) Name Form definitions that define primitive naming relations
      for structural object classes;

   b) DIT Structure Rule definitions that define the names that
      entries may have and the ways in which the entries may be
      related to one another in the DIT;

   c) DIT Content Rule definitions that extend the specification of
      allowable attributes for entries beyond those indicated by the
      structural object classes of the entries;

   d) Object Class definitions that define the basic set of mandatory
      and optional attributes that shall be present, and may be
      present, respectively, in an entry of a given class, and which
      indicate the kind of object class that is being defined;

   e) Attribute Type definitions that identify the object identifier
      by which an attribute is known, its syntax, associated matching
      rules, whether it is an operational attribute and if so its
      type, whether it is a collective attribute, whether it is
      permitted to have multiple values and whether or not it is
      derived from another attribute type;

   f) Matching Rule definitions that define matching rules.

   And in LDAP:

   g) LDAP Syntaxes definitions that define encodings used in LDAP.

4.1. Schema Definitions

   Schema definitions in this section are described using ABNF and rely
   on the common productions specified in Section 1.2 as well as these:

       noidlen = numericoid [LCURLY len RCURLY]

       len = number

       oids = oid / (LPAREN WSP oidlist WSP RPAREN)

       oidlist = oid *(WSP DOLLAR WSP oid)

       extensions = *(SP xstring SP qdstrings)

       xstring = X HYPHEN 1*(ALPHA / HYPHEN / USCORE)

       qdescrs = qdescr / (LPAREN WSP qdescrlist WSP RPAREN)

```
      qdescrlist = [ qdescr *( SP qdescr ) ]

      qdescr = SQUOTE descr SQUOTE

      qdstrings = qdstring / ( LPAREN WSP qdstringlist WSP RPAREN )

      qdstringlist = [ qdstring *( SP qdstring ) ]

      qdstring = SQUOTE dstring SQUOTE

      dstring = 1*( QS / QQ / QUTF8 )   ; escaped UTF8 string

      QQ =  ESC %x32 %x37 ; "\27"

      QS =  ESC %x35 ( %x43 / %x63 ) ; "\5C" / "\5c"

      ; Any UTF-8 encoded UCS character
      ; except %x27 ("'") and %x5C ("\")
      QUTF8     = QUTF1 / UTFMB

      ; Any ASCII character except %x27 ("'") and %x5C ("\")
      QUTF1     = %x00-26 / %x28-5B / %x5D-7F
```

   Schema definitions in this section also share a number of common
   terms.

   The NAME field provides a set of short names (descriptors) which are
   be used as aliases for the OID.

   The DESC field optionally allows a descriptive string to be provided
   by the directory administrator and/or implementor. While
   specifications may suggest a descriptive string, there is no
   requirement that the suggested (or any) descriptive string be used.

   Implementors should note that future versions of this document may
   expand these definitions to include additional terms. Terms whose
   identifier begins with "X-" are reserved for private experiments, and
   MUST be followed by <space> and <qdstrings> tokens.

4.1.1. Object Class Definitions

   Object Class definitions are written according to the ABNF:

     ObjectClassDescription = LPAREN WSP
         numericoid   ; object identifier
         [SP "NAME" SP qdescrs]   ; short names (descriptors)
         [SP "DESC" SP qdstring] ; description
```

```
        [ SP "OBSOLETE" ]          ; not active
        [ SP "SUP" SP oids ]       ; superior object classes
        [ SP kind ]                ; kind of class
        [ SP "MUST" SP oids ]      ; attribute types
        [ SP "MAY" SP oids ]       ; attribute types
        extensions WSP RPAREN

    kind = "ABSTRACT" / "STRUCTURAL" / "AUXILIARY"
```

  where:
    <numericoid> is object identifier assigned to this object class;
    NAME <qdescrs> are short names (descriptors) identifying this object
        class;
    DESC <qdstring> is a short descriptive string;
    OBSOLETE indicates this object class is not active;
    SUP <oids> specifies the direct superclasses of this object class;
    the kind of object class is indicated by one of ABSTRACT,
        STRUCTURAL, or AUXILIARY, default is STRUCTURAL;
    MUST and MAY specify the sets of required and allowed attribute
        types, respectively; and
    <extensions> describe extensions.

4.1.2. Attribute Types

  Attribute Type definitions are written according to the ABNF:

```
    AttributeTypeDescription = LPAREN WSP
        numericoid                 ; object identifier
        [ SP "NAME" SP qdescrs ]   ; short names (descriptors)
        [ SP "DESC" SP qdstring ]  ; description
        [ SP "OBSOLETE" ]          ; not active
        [ SP "SUP" SP oid ]        ; subtype
        [ SP "EQUALITY" SP oid ]   ; equality matching rule
        [ SP "ORDERING" SP oid ]   ; ordering matching rule
        [ SP "SUBSTR" SP oid ]     ; substrings matching rule
        [ SP "SYNTAX" SP noidlen ] ; value syntax
        [ SP "SINGLE-VALUE" ]      ; single-value
        [ SP "COLLECTIVE" ]        ; collective
        [ SP "NO-USER-MODIFICATION" ] ; not user modifiable
        [ SP "USAGE" SP usage ]    ; usage
        extensions WSP RPAREN      ; extensions

    usage = "userApplications"     /
            "directoryOperation"   / ; directory operational
            "distributedOperation" / ; DSA-shared operational
            "dSAOperation"           ; DSA-specific operational
```

  where:
    <numericoid> is object identifier assigned to this attribute type;
    NAME <qdescrs> are short names (descriptors) identifying this
        attribute type;
    DESC <qdstring> is a short descriptive string;
    OBSOLETE indicates this attribute type is not active;
    SUP oid specifies the direct supertype of this type;
    EQUALITY, ORDERING, SUBSTRING provide the oid of the equality,
        ordering, and substrings matching rules, respectively;
    SYNTAX identifies value syntax by object identifier and may suggest
        a minimum upper bound;
    COLLECTIVE indicates this attribute type is collective [X.501];
    NO-USER-MODIFICATION indicates this attribute type is not user
        modifiable;
    USAGE indicates the application of this attribute type; and
    <extensions> describe extensions.

  Each attribute type description must contain at least one of the SUP
  or SYNTAX fields.

  Usage of userApplications, the default, indicates that attributes of
  this type represent user information. That is, they are user
  attributes.

  COLLECTIVE requires usage userApplications. Use of collective
  attribute types in LDAP is not discussed in this technical
  specification.

  A usage of directoryOperation, distributedOperation, or dSAOperation
  indicates that attributes of this type represent operational and/or
  administrative information. That is, they are operational attributes.

  directoryOperation usage indicates that the attribute of this type is
  a directory operational attribute. distributedOperation usage
  indicates that the attribute of this DSA-shared usage operational
  attribute. dSAOperation usage indicates that the attribute of this
  type is a DSA-specific operational attribute.

  NO-USER-MODIFICATION requires an operational usage.

  Note that the <AttributeTypeDescription> does not list the matching
  rules which can can be used with that attribute type in an
  extensibleMatch search filter. This is done using the
  'matchingRuleUse' attribute described in Section 4.1.3.

  This document refines the schema description of X.501 by requiring
  that the SYNTAX field in an <AttributeTypeDescription> be a string
  representation of an object identifier for the LDAP string syntax

   definition with an optional indication of the suggested minimum bound
   of a value of this attribute.

   A suggested minimum upper bound on the number of characters in a value
   with a string-based syntax, or the number of bytes in a value for all
   other syntaxes, may be indicated by appending this bound count inside
   of curly braces following the syntax's OBJECT IDENTIFIER in an
   Attribute Type Description. This bound is not part of the syntax name
   itself. For instance, "1.3.6.4.1.1466.0{64}" suggests that server
   implementations should allow a string to be 64 characters long,
   although they may allow longer strings. Note that a single character
   of the Directory String syntax may be encoded in more than one octet
   since UTF-8 is a variable-length encoding.

4.1.3. Matching Rules

   Matching rules are used by servers to compare attribute values against
   assertion values when performing Search and Compare operations. They
   are also used to identify the value to be added or deleted when
   modifying entries, and are used when comparing a purported
   distinguished name with the name of an entry.

   A matching rule specifies the syntax of the assertion value.

   Each matching rule is identified by an object identifier (OID) and,
   optionally, one or more short names (descriptors).

   Matching rule definitions are written according to the ABNF:

     MatchingRuleDescription = LPAREN WSP
         numericoid   ; object identifier
         [SP "NAME" SP qdescrs]   ; short names (descriptors)
         [SP "DESC" SP qdstring] ; description
         [SP "OBSOLETE"]   ; not active
         SP "SYNTAX" SP numericoid ; oid corrected to numericoid
         extensions WSP RPAREN   ; extensions

   where:
     <numericoid> is object identifier assigned to this matching rule;
     NAME <qdescrs> are short names (descriptors) identifying this
         matching rule;
     DESC <qdstring> is a short descriptive string;
     OBSOLETE indicates this matching rule is not active;
     SYNTAX identifies the assertion syntax by object identifier; and
     <extensions> describe extensions.

   A matching rule use lists the attributes which are suitable for use

with an extensible matching rule.

Matching rule use descriptions (see Section 4.1.3) are written
according to the following ABNF:

```
MatchingRuleUseDescription = LPAREN WSP
    numericoid                 ; object identifier
    [ SP "NAME" SP qdescrs ]   ; short names (descriptors)
    [ SP "DESC" SP qdstring ]  ; description
    [ SP "OBSOLETE" ]          ; not active
    SP "APPLIES" SP oids       ; attribute types
    extensions WSP RPAREN      ; extensions
```

where:
  <numericoid> is the object identifier of the matching rule
      associated with this matching rule use description;
  NAME <qdescrs> are short names (descriptors) identifying this
      matching rule use;
  DESC <qdstring> is a short descriptive string;
  OBSOLETE indicates this matching rule use is not active;
  APPLIES provides a list of attribute types the matching rule applies
      to; and
  <extensions> describe extensions.

4.1.4. LDAP Syntaxes

LDAP Syntaxes of (attribute and assertion) values are described in
terms of ASN.1 [X.680] and, optionally, have an octet string encoding
known as the LDAP-specific encoding. Commonly, the LDAP-specific
encoding is constrained to string of Universal Character Set (UCS)
[ISO10646] characters in UTF-8 [RFC2279] form.

Each LDAP syntax is identified by an object identifier (OID). These
are not intended to be displayed to users.

LDAP syntax definitions are written according to the ABNF:

```
SyntaxDescription = LPAREN WSP
    numericoid                 ; object identifier
    [ SP "DESC" SP qdstring ]  ; description
    extensions WSP RPAREN      ; extensions
```

where:
  <numericoid> is object identifier assigned to this LDAP syntax;
  DESC <qdstring> is a short descriptive string; and
  <extensions> describe extensions.

4.1.5. **DIT Content Rules**

   A DIT content rule is a "rule governing the content of entries of a
   particular structural object class" [X.501].

   For DIT entries of a particular structural object class, a DIT content
   rule specifies which auxiliary object classes the entries are allowed
   to belong to and which additional attributes (by type) are required,
   allowed or not allowed to appear in the entries.

   The list of precluded attributes cannot include any attribute listed
   as mandatory in rule, the structural object class, or any of the
   allowed auxiliary object classes.

   Each content rule is identified by the object identifier, as well as
   any short names (descriptors), of the structural object class it
   applies to.

   An entry may only belong to auxiliary object classes listed in the
   governing content rule.

   An entry must contain all attributes required by the object classes
   the entry belongs to as well as all attributed required by the
   governing content rule.

   An entry may contain any non-precluded attributes allowed by the
   object classes the entry belongs to as well as all attributes allowed
   by the governing content rule.

   An entry cannot include any attribute precluded by the governing
   content rule.

   An entry is governed by (if present and active in the subschema) the
   DIT content rule which applies to the structural object class of the
   entry (see Section 2.4.2). If no active rule is present for the
   entry's structural object class, the entry's content is governed by
   the structural object class (and possibly other aspects of user and
   system schema).

   DIT content rule descriptions are written according to the ABNF:

      DITContentRuleDescription = LPAREN WSP
          numericoid   ; object identifier
          [SP "NAME" SP qdescrs]   ; short names (descriptors)
          [SP "DESC" SP qdstring] ; description
          [SP "OBSOLETE"]   ; not active
          [SP "AUX" SP oids]   ; auxiliary object classes
          [SP "MUST" SP oids]   ; attribute types

```
      [ SP "MAY" SP oids ]        ; attribute types
      [ SP "NOT" SP oids ]        ; attribute types
      extensions WSP RPAREN       ; extensions
```

   where:
     <numericoid> is the object identifier of the structural object class
         associated with this DIT content rule;
     NAME <qdescrs> are short names (descriptors) identifying this DIT
         content rule;
     DESC <qdstring> is a short descriptive string;
     OBSOLETE indicates this DIT content rule use is not active;
     AUX specifies a list of auxiliary object classes which entries
         subject to this DIT content rule may belong to;
     MUST, MAY, and NOT specify lists of attribute types which are
         required, allowed, or precluded, respectively, from appearing in
         entries subject to this DIT content rule; and
     <extensions> describe extensions.

4.1.6. **DIT Structure Rules and Name Forms**

  It is sometimes desirable to regulate where object entries can be
  placed in the DIT and how they can be named based upon their
  structural object class.

  A DIT structure rule is a "rule governing the structure of the DIT by
  specifying a permitted superior to subordinate entry relationship. A
  structure rule relates a name form, and therefore a structural object
  class, to superior structure rules. This permits entries of the
  structural object class identified by the name form to exist in the
  DIT as subordinates to entries governed by the indicated superior
  structure rules" [X.501].

  A name form "specifies a permissible RDN for entries of a particular
  structural object class. A name form identifies a named object class
  and one or more attribute types to be used for naming (i.e. for the
  RDN). Name forms are primitive pieces of specification used in the
  definition of DIT structure rules" [X.501].

  Each name form indicates the structural object class to be named, a
  set of required attribute types, and a set of allowed attributes
  types. A particular attribute type cannot be listed in both sets.

  Entries governed by the form must be named using a value from each
  required attribute type and zero or more values from the allowed
  attribute types.

  Each name form is identified by an object identifier (OID) and,

optionally, one or more short names (descriptors).

DIT structure rule descriptions are written according to the ABNF:

```
DITStructureRuleDescription = LPAREN WSP
    ruleid                     ; rule identifier
    [ SP "NAME" SP qdescrs ]   ; short names (descriptors)
    [ SP "DESC" SP qdstring ]  ; description
    [ SP "OBSOLETE" ]          ; not active
    SP "FORM" SP oid           ; NameForm
    [ SP "SUP" ruleids ]       ; superior rules
    extensions WSP RPAREN      ; extensions

ruleids = ruleid / ( LPAREN WSP ruleidlist WSP RPAREN )

ruleidlist = ruleid *( SP ruleid )

ruleid = number
```

where:
  <ruleid> is the rule identifier of this DIT structure rule;
  NAME <qdescrs> are short names (descriptors) identifying this DIT
      structure rule;
  DESC <qdstring> is a short descriptive string;
  OBSOLETE indicates this DIT structure rule use is not active;
  FORM is specifies the name form associated with this DIT structure
      rule;
  SUP identifies superior rules (by rule id); and
  <extensions> describe extensions.

Name form descriptions are written according to the ABNF:

```
NameFormDescription = LPAREN WSP
    numericoid                 ; object identifier
    [ SP "NAME" SP qdescrs ]   ; short names (descriptors)
    [ SP "DESC" SP qdstring ]  ; description
    [ SP "OBSOLETE" ]          ; not active
    SP "OC" SP oid             ; structural object class
    SP "MUST" SP oids          ; attribute types
    [ SP "MAY" SP oids ]       ; attribute types
    extensions WSP RPAREN      ; extensions
```

where:
  <numericoid> is object identifier assigned to this name form;
  NAME <qdescrs> are short names (descriptors) identifying this name
      form;
  DESC <qdstring> is a short descriptive string;
  OBSOLETE indicates this name form is not active;

   OC identifies the structural object class this rule applies to,
   MUST and MAY specify the sets of required and allowed, respectively,
       naming attributes for this name form; and
   <extensions> describe extensions.

4.2. Subschema Subentries

  Subschema (sub)entries are used for administering information about
  the directory schema. A single subschema (sub)entry contains all
  schema definitions (see Section 4.1) used by entries in a particular
  part of the directory tree.

  Servers which follow X.500(93) models SHOULD implement subschema using
  the X.500 subschema mechanisms (as detailed in Section 12 of [X.501]),
  and so these are not ordinary object entries but subentries (see
  Section 3.2). LDAP clients SHOULD NOT assume that servers implement
  any of the other aspects of X.500 subschema.

  Servers MAY allow subschema modification. Procedures for subschema
  modification are discussed in Section 14.5 of [X.501].

  A server which masters entries and permits clients to modify these
  entries MUST implement and provide access to these subschema
  (sub)entries including providing a 'subschemaSubentry' attribute in
  each modifiable entry. This so clients may discover the attributes
  and object classes which are permitted to be present. It is strongly
  RECOMMENDED that all other servers implement this as well.

  The value of the 'subschemaSubentry' attribute is the name of the
  subschema (sub)entry holding the subschema controlling the entry.

      (2.5.18.10 NAME 'subschemaSubentry'
        EQUALITY distinguishedNameMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
        NO-USER-MODIFICATION SINGLE-VALUE
        USAGE directoryOperation)

  The 'distinguishedNameMatch' matching rule and the DistinguishedName
  (1.3.6.1.4.1.1466.115.121.1.12) syntax are defined in [Syntaxes].

  Subschema is held in (sub)entries belonging to the subschema auxiliary
  object class.

      (2.5.20.1 NAME 'subschema' AUXILIARY
        MAY (dITStructureRules $ nameForms $ ditContentRules $
          objectClasses $ attributeTypes $ matchingRules $
          matchingRuleUse))

The 'ldapSyntaxes' operational attribute may also be present in
subschema entries.

Servers MAY provide additional attributes (described in other
documents) in subschema (sub)entries.

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

The following subsections provide attribute type definitions for each
of schema definition attribute types.

4.2.1. 'objectClasses'

This attribute holds definitions of object classes supported in the
subschema.

```
    ( 2.5.21.6 NAME 'objectClasses'
      EQUALITY objectIdentifierFirstComponentMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.37
      USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
ObjectClassDescription (1.3.6.1.4.1.1466.115.121.1.37) syntax are
defined in [Syntaxes].

4.2.2. 'attributeTypes'

This attribute holds definitions of attribute types supported in the
subschema.

```
    ( 2.5.21.5 NAME 'attributeTypes'
      EQUALITY objectIdentifierFirstComponentMatch
      SYNTAX 1.3.6.1.4.1.1466.115.121.1.3
      USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
AttributeTypeDescription (1.3.6.1.4.1.1466.115.121.1.3) syntax are
defined in [Syntaxes].

4.2.3. 'matchingRules'

This attribute holds definitions of matching rules supported in the
subschema.

```
   ( 2.5.21.4 NAME 'matchingRules'
     EQUALITY objectIdentifierFirstComponentMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.30
     USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
MatchingRuleDescription (1.3.6.1.4.1.1466.115.121.1.30) syntax are
defined in [Syntaxes].

4.2.4 'matchingRuleUse'

This attribute holds definitions of matching rule uses supported in
the subschema.

```
   ( 2.5.21.8 NAME 'matchingRuleUse'
     EQUALITY objectIdentifierFirstComponentMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.31
     USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
MatchingRuleUseDescription (1.3.6.1.4.1.1466.115.121.1.31) syntax are
defined in [Syntaxes].

4.2.5. 'ldapSyntaxes'

This attribute holds definitions of LDAP syntaxes supported in the
subschema.

```
   ( 1.3.6.1.4.1.1466.101.120.16 NAME 'ldapSyntaxes'
     EQUALITY objectIdentifierFirstComponentMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.54
     USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
SyntaxDescription (1.3.6.1.4.1.1466.115.121.1.54) syntax are defined
in [Syntaxes].

4.2.6. 'dITContentRules'

This attribute lists DIT Content Rules which are in force.

```
   ( 2.5.21.2 NAME 'dITContentRules'
     EQUALITY objectIdentifierFirstComponentMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.16
     USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
DITContentRuleDescription (1.3.6.1.4.1.1466.115.121.1.16) syntax are
defined in [Syntaxes].

4.2.7. 'dITStructureRules'

This attribute lists DIT Structure Rules which are in force.

```
( 2.5.21.1 NAME 'dITStructureRules'
  EQUALITY integerFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.17
  USAGE directoryOperation )
```

The 'integerFirstComponentMatch' matching rule and the
DITStructureRuleDescription (1.3.6.1.4.1.1466.115.121.1.17) syntax are
defined in [Syntaxes].

4.2.8 'nameForms'

This attribute lists Name Forms which are in force.

```
( 2.5.21.7 NAME 'nameForms'
  EQUALITY objectIdentifierFirstComponentMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.35
  USAGE directoryOperation )
```

The 'objectIdentifierFirstComponentMatch' matching rule and the
NameFormDescription (1.3.6.1.4.1.1466.115.121.1.35) syntax are defined
in [Syntaxes].

4.3. 'extensibleObject' object class

The 'extensibleObject auxiliary object class allows entries belong to
it to hold any attribute type. The set of allowed attributes of this
class is implicitly the set of all user attributes.

```
( 1.3.6.1.4.1.1466.101.120.111 NAME 'extensibleObject'
  SUP top AUXILIARY )
```

The mandatory attributes of the other object classes of this entry are
still required to be present and any precluded attributes are still
not allowed to be present.

Note that not all servers will implement this object class, and those
which do not will reject requests to add entries which contain this

object class, or modify an entry to add this object class.

4.4. Subschema Discovery

To discover the DN of the subschema (sub)entry holding the subschema
controlling a particular entry, a client reads that entry's
'subschemaSubentry' operational attribute. To read schema attributes
from the subschema (sub)entry, clients MUST issue a base object search
where the filter is "(objectClass=subschema)" [Filters] and the list
of attributes includes the names of the desired schema attributes (as
they are operational). This filter allows LDAP servers which gateway
to X.500 to detect that subentry information is being requested.

5. DSA (Server) Informational Model

The LDAP protocol assumes there are one or more servers which jointly
provide access to a Directory Information Tree (DIT).

As defined in [X.501]:

    context prefix: The sequence of RDNs leading from the Root of the
        DIT to the initial vertex of a naming context; corresponds to
        the distinguished name of that vertex.

    DIB fragment: The portion of the DIB that is held by one master
        DSA, comprising one or more naming contexts.

    naming context: A subtree of entries held in a single master DSA.

That is, a naming context is the largest collection of entries,
starting at an entry that is mastered by a particular server, and
including all its subordinates and their subordinates, down to the
entries which are mastered by different servers. The context prefix
is the name of the initial entry.

The root of the DIT is a DSA-specific Entry (DSE) and not part of any
naming context (or any subtree); each server has different attribute
values in the root DSE.

5.1. Server-specific Data Requirements

An LDAP server MUST provide information about itself and other
information that is specific to each server. This is represented as a
group of attributes located in the root DSE (DSA-Specific Entry),
which is named with the zero-length LDAPDN. These attributes are

   retrievable, subject to access control and other restrictions, if a
   client performs a base object search of the root with the filter
   "(objectClass=*)" [Filters] requesting the desired attributes. It is
   noted that root DSE attributes are operational, and like other
   operational attributes, are not returned in search requests unless
   requested by name.

   The root DSE SHALL NOT be included if the client performs a subtree
   search starting from the root.

   Servers may allow clients to modify attributes of the root DSE where
   appropriate.

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

     - altServer: alternative servers;

     - namingContexts: naming contexts;

     - supportedControl: recognized LDAP controls;

     - supportedExtension: recognized LDAP extended operations;

     - supportedLDAPVersion: LDAP versions supported; and

     - supportedSASLMechanisms: recognized SASL mechnanisms.

   The values of these attributes provided may depend on session specific
   and other factors. For example, a server supporting the SASL EXTERNAL
   mechanism might only list "EXTERNAL" when the client's identity has
   been established by a lower level. See [AuthMeth].

   The root DSE may also include a 'subschemaSubentry' attribute. If so,
   it refers to the subschema (sub)entry holding schema controlling
   attributes of the root DSE. Client SHOULD NOT assume that the
   subschema (sub)entry controlling the root DSE controls any entry held
   by the server. General subschema discovery procedures are provided in
   Section 4.4.

5.1.1. 'altServer'

   The 'altServer' attribute lists URLs referring to alternative servers
   which may be contacted when this server becomes unavailable. If the
   server does not know of any other servers which could be used this
   attribute will be absent. Clients may cache this information in case
   their preferred LDAP server later becomes unavailable.

```
( 1.3.6.1.4.1.1466.101.120.6 NAME 'altServer'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
  USAGE dSAOperation )
```

The IA5String (1.3.6.1.4.1.1466.115.121.1.26) syntax is defined in
[Syntaxes].

5.1.2. 'namingContexts'

The 'namingContexts' attribute lists the context prefixes of the
naming contexts the server masters or shadows (in part or in whole).
If the server does not master or shadow any information (e.g. it is an
LDAP gateway to a public X.500 directory) this attribute will be
absent. If the server believes it masters or shadows the entire
directory, the attribute will have a single value, and that value will
be the empty string (indicating the null DN of the root). This
attribute allows a client to choose suitable base objects for
searching when it has contacted a server.

```
( 1.3.6.1.4.1.1466.101.120.5 NAME 'namingContexts'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
  USAGE dSAOperation )
```

The DistinguishedName (1.3.6.1.4.1.1466.115.121.1.12) syntax is
defined in [Syntaxes].

5.1.3. 'supportedControl'

The 'supportedControl' attribute lists object identifiers identifying
the request controls the server supports. If the server does not
support any request controls, this attribute will be absent.

Object identifiers identifying response controls need not be listed.

Procedures for registering object identifiers used to discovery of
protocol mechanisms are detailed in BCP 64 [RFC3383].

```
( 1.3.6.1.4.1.1466.101.120.13 NAME 'supportedControl'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
  USAGE dSAOperation )
```

The OBJECT IDENTIFIER (1.3.6.1.4.1.1466.115.121.1.38) syntax is
defined in [Syntaxes].

5.1.4. 'supportedExtension'

The 'supportedExtension' attribute lists object identifiers
identifying the extended operations which the server supports. If the
server does not support any extended operations, this attribute will
be absent.

An extended operation comprises a ExtendedRequest, possibly other PDUs
defined by extension, and an ExtendedResponse [Protocol]. The object
identifier assigned to the ExtendedRequest is used to identify the
extended operation. Other object identifiers associated with the
extended operation need not be listed.

Procedures for registering object identifiers used to discovery of
protocol mechanisms are detailed in BCP 64 [RFC3383].

        (1.3.6.1.4.1.1466.101.120.7 NAME 'supportedExtension'
          SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
          USAGE dSAOperation)

The OBJECT IDENTIFIER (1.3.6.1.4.1.1466.115.121.1.38) syntax is
defined in [Syntaxes].

5.1.5. 'supportedLDAPVersion'

The 'supportedLDAPVersion' attribute lists the versions of LDAP which
the server supports.

        (1.3.6.1.4.1.1466.101.120.15 NAME 'supportedLDAPVersion'
          SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
          USAGE dSAOperation)

The INTEGER (1.3.6.1.4.1.1466.115.121.1.27) syntax are defined in
[Syntaxes].

5.1.6. 'supportedSASLMechanisms'

The 'supportedSASLMechanisms' attribute lists the SASL mechanisms
[RFC2222] which the server recognizes. The contents of this attribute
may depend on the current session state. If the server does not
support any SASL mechanisms this attribute will not be present.

        (1.3.6.1.4.1.1466.101.120.14 NAME 'supportedSASLMechanisms'
          SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
          USAGE dSAOperation)

The Directory String (1.3.6.1.4.1.1466.115.121.1.15) syntax is defined
in [Syntaxes].

6. Other Considerations

6.1. Preservation of User Information

   Syntaxes may be defined which have specific value and/or value form
   (representation) preservation requirements. For example, a syntax
   containing digitally signed data can mandate the server preserve both
   the value and form of value presented to ensure signature is not
   invalidated.

   Where such requirements have not be explicitly stated, servers SHOULD
   preserve the value of user information but MAY return the value in a
   different form. And where a server is unable (or unwilling) to
   preserve the value of user information, the server SHALL ensure that
   an equivalent value (per Section 2.3) is returned.

6.2. Short Names

   Short names, also known as descriptors, are used as more readable
   aliases for object identifiers and are used to identify various schema
   elements.

   The same short name might have different meaning in different
   subschemas and, within a particular subschema, the same short name
   might refer to different object identifiers each identifying a
   different kind of schema element.

   Implementations MUST be prepared that the same short name might be
   used in a subschema to refer to the different kinds of schema
   elements. That is, there might be an object class 'x-fubar' and an
   attribute type 'x-fubar' in a subschema.

   Implementations MUST be prepared that the same short name might be
   used in the different subschemas to refer to the different schema
   elements. That is, there might be two matching rules 'x-fubar', each
   in different subschemas.

   Procedures for registering short names (descriptors) are detailed in
   BCP 64 [RFC3383bis].

   [[The remainder of this subsection will be included a subsequent
   revision of RFC 3383.]]

   To avoid interoperability problems, the following additional
   considerations are stated:

      Descriptors used to identify various schema SHOULD be registered

    unless in private-use name space (e.g., they begin with "x-").
    Descriptors defined in RFCs MUST be registered.

    While the protocol allows the same descriptor to refer to
    different object identifiers in certain cases and the registry
    supports multiple registrations of the same descriptor (each
    indicating a different kind of schema element and different object
    identifier), multiple registrations of the same descriptor are to
    be avoided. All such registration requests require Expert Review.

6.3. Cache and Shadowing

  Some servers may hold cache or shadow copies of entries, which can be
  used to answer search and comparison queries, but will return
  referrals or contact other servers if modification operations are
  requested. Servers that perform shadowing or caching MUST ensure that
  they do not violate any access control constraints placed on the data
  by the originating server.

7. Implementation Guidelines

7.1 Server Guidelines

  Servers MUST recognize all attribute types and object classes defined
  in this document but, unless stated otherwise, need not support the
  associated functionality. Servers SHOULD recognize all the names of
  attribute types and object classes defined in Section 3 and 4,
  respectively, of [Schema].

  Servers MUST ensure that entries conform to user and system schema
  rules or other data model constraints.

  Servers MAY support DIT Content Rules, DIT Structure Rules, and/or
  Name Forms features.

  Servers MAY support alias entries.

  Servers MAY support subentries. If so, they MUST do so in accordance
  with [X.501]. Servers which do not support subentries SHOULD use
  object entries to mimic subentries as detailed in Section 3.2.

  Servers MAY support the 'extensibleObject' object class.

  Servers MAY implement additional schema elements. Servers SHOULD
  provide the definitions of all schema elements they support in
  subschema (sub)entries.

7.2 Client Guidelines

  Clients MUST NOT display nor attempt to decode as ASN.1, a value if
  its syntax is not known. The implementation may attempt to discover
  the subschema of the source entry, and retrieve the values of
  'attributeTypes' from the subschema (sub)entry.

  Clients MUST NOT assume the LDAP-specific string encoding is
  restricted to a UTF-8 encoded string of UCS characters or any
  particular subset of particular subset of UCS (such as a printable
  subset) unless such restriction is explicitly stated.

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

8. Security Considerations

  Attributes of directory entries are used to provide descriptive
  information about the real-world objects they represent, which can be
  people, organizations or devices. Most countries have privacy laws
  regarding the publication of information about people.

  General security considerations for accessing directory information
  with LDAP are discussed in [Protocol] and [AuthMeth].

9. IANA Considerations

  It is requested that the Internet Assigned Numbers Authority (IANA)
  update the LDAP descriptors registry as indicated the following
  template:

      Subject: Request for LDAP Descriptor Registration Update
      Descriptor (short name): see comment
      Object Identifier: see comment
      Person & email address to contact for further information:
          Kurt Zeilenga <kurt@OpenLDAP.org>
      Usage: see comment
      Specification: RFC XXXX
      Author/Change Controller: IESG
      Comments:

      The following descriptors (short names) should be updated to refer
      to RFC XXXX.

        NAME   Type OID
        -----------------------   ---- ----------------

```
        alias                         O 2.5.6.1
        aliasedEntryName              A 2.5.4.1
        aliasedObjectName             A 2.5.4.1
        altServer                     A 1.3.6.1.4.1.1466.101.120.6
        attributeTypes                A 2.5.21.5
        createTimestamp               A 2.5.18.1
        creatorsName                  A 2.5.18.3
        dITContentRules               A 2.5.21.2
        dITStructureRules             A 2.5.21.1
        extensibleObject              O 1.3.6.1.4.1.1466.101.120.111
        ldapSyntaxes                  A 1.3.6.1.4.1.1466.101.120.16
        matchingRuleUse               A 2.5.21.8
        matchingRules                 A 2.5.21.4
        modifiersName                 A 2.5.18.4
        modifyTimestamp               A 2.5.18.2
        nameForms                     A 2.5.21.7
        namingContexts                A 1.3.6.1.4.1.1466.101.120.5
        objectClass                   A 2.5.4.0
        objectClasses                 A 2.5.21.6
        subschema                     O 2.5.20.1
        subschemaSubentry             A 2.5.18.10
        supportedControl              A 1.3.6.1.4.1.1466.101.120.13
        supportedExtension            A 1.3.6.1.4.1.1466.101.120.7
        supportedLDAPVersion          A 1.3.6.1.4.1.1466.101.120.15
        supportedSASLMechanisms       A 1.3.6.1.4.1.1466.101.120.14
        top                           O 2.5.6.0
```

10. Acknowledgments

This document is based in part on RFC 2251 by M. Wahl, T. Howes, and
S. Kille; RFC 2252 by M. Wahl, A. Coulbeck, T. Howes, S. Kille; and
RFC 2556 by M. Wahl, all products of the IETF Access, Searching and
Indexing of Directories (ASID) Working Group. This document is also
based in part on "The Directory: Models" [X.501], a product of the
International Telephone Union (ITU).

11. Author's Address

Kurt Zeilenga
E-mail: <kurt@openldap.org>

12. References

12.1. Normative References

   [RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646",
              RFC 2279, January 1998.

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

   [RFC2234] Crocker, D., and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", RFC 2234, November 1997.

   [RFC3383] K. Zeilenga, "IANA Considerations for LDAP", BCP 64 (also
              RFC 3383), September 2002.

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

   [Protocol] J. Sermersheim (editor), "LDAP: The Protocol",
              draft-ietf-ldapbis-protocol-xx.txt, a work in progress.

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

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

   [Filters] M. Smith (editor), LDAPbis WG, "LDAP: String Representation
              of Search Filters", draft-ietf-ldapbis-filter-xx.txt, a
              work in progress.

   [LDAPURL] M. Smith (editor), "LDAP: Uniform Resource Locator",
              draft-ietf-ldapbis-url-xx.txt, a work in progress.

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

   [Schema]   K. Dally (editor), "LDAP: User Schema",
              draft-ietf-ldapbis-user-schema-xx.txt, a work in progress.

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

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

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

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

12.2. Informative References

   None.

Appendix A.   Changes

   This appendix is non-normative.

   This document amounts to nearly a complete rewrite of portions of RFC
   2251, RFC 2252, and RFC 2256. This rewrite was undertaken to improve
   overall clarity of technical specification. This appendix provides a
   summary of substantive changes made to the portions of these documents
   incorporated into this document. Readers should consult [Roadmap],
   [Protocol], [Syntaxes], and [Schema] for summaries of remaining
   portions of these documents.

A.1 Changes to RFC 2251

   This document incorporates from RFC 2251 sections 3.2 and 3.4,
   portions of Section 4 and 6 as summarized below.

A.1.1 Section 3.2 of RFC 2251

   Section 3.2 of RFC 2251 provided a brief introduction to the X.500
   data model, as used by LDAP. The previous specification relied on
   [X.501] but lacked clarity in how X.500 models are adapted for use by
   LDAP. This document describes the X.500 data models, as used by LDAP
   in greater detail, especially in areas where the models require
   adaptation is needed.

   Section 3.2.1 of RFC 2251 described an attribute as "a type with one
   or more associated values." In LDAP, an attribute is better described
   as an attribute description, a type with zero or more options, and one
   or more associated values.

   Section 3.2.2 of RFC 2251 mandated that subschema subentries contain
   objectClasses and attributeTypes attributes, yet X.500(93) treats
   these attributes as optional. While generally all implementations
   that support X.500(93) subschema mechanisms will provide both of these
   attributes, it is not absolutely required for interoperability that
   all servers do. The mandate was removed for consistency with

X.500(93).   The subschema discovery mechanism was also clarified to
indicate that subschema controlling an entry is obtained by reading
the (sub)entry referred to by that entry's 'subschemaSubentry'
attribute.

A.1.2 Section 3.4 of RFC 2251

Section 3.4 of RFC 2251 provided "Server-specific Data Requirements".
This material, with changes, was incorporated in Section 5.1 of this
document.

Changes:

- Clarify that attributes of the root DSE are subject to "other
  restrictions" in addition to acccess controls.

- Clarify that only recognized extended requests need to be enumerated
  'supportedExtension'.

- Clarify that only recognized request controls need to be enumerated
  'supportedControl'.

- Clarify that root DSE attributes are operational and, like other
  operational attributes, will not be returned in search requests
  unless requested by name.

- Clarify that not all root DSE attributes are user modifiable.

- Remove inconsistent text regarding handling of the
  'subschemaSubentry' attribute within the root DSE. The previous
  specification stated that the 'subschemaSubentry' attribute held in
  the root DSE referred to "subschema entries (or subentries) known by
  this server." This is inconsistent with the attribute intended use
  as well as its formal definition as a single valued attribute
  [X.501]. It is also noted that a simple (possibly incomplete) list
  of subschema (sub)entries is not terrible useful. This document (in
  section 5.1) specifies that the 'subschemaSubentry' attribute of the
  root DSE refers to the subschema controlling the root DSE. It is
  noted that the general subschema discovery mechanism remains
  available (see Section 4.4 of this document).

A.1.2 Section 4 of RFC 2251

Portions of Section 4 of RFC 2251 detailing aspects of the information
model used by LDAP were incorporated in this document, including:

- Restriction of distinguished values to attributes whose descriptions
  have no options (from Section 4.1.3).

- Data model aspects of Attribute Types (from Section 4.1.4),
  Attribute Descriptions (from 4.1.4), Attribute (from 4.1.8),
  Matching Rule Identifier (from 4.1.9).

- User schema requirements (from Section 4.1.6, 4.5.1, and 4.7).

A.1.3 Section 6 of RFC 2251

The Section 6.1 and the second paragraph of Section 6.2 of RFC 2251
where incorporated into this document.

A.2 Changes to RFC 2252

This document incorporates Sections 4, 5 and 7 from RFC 2252.

A.2.1 Section 4 of RFC 2252

The specification was updated to use Augmented BNF [RFC2234]. The
string representation of an OBJECT IDENTIFIER was tighten to
disallow leading zeros as described in RFC 2252 text.

The <descr> syntax was changed to disallow semicolon (U+0003B)
characters to appear to be consistent its natural language
specification "descr is the syntactic representation of an object
descriptor, which consists of letters and digits, starting with a
letter." In a related change, the statement "an
AttributeDescription can be used as the value in a NAME part of an
AttributeTypeDescription" was deleted. RFC 2252 provided no
specification as to the semantics of attribute options appearing in
NAME fields.

The ABNF for a quoted string (qdstring) was updated to reflect
support for the escaping mechanism described in 4.3 of RFC 2252.

A.2.2 Section 5 of RFC 2252

Definitions of operational attributes provided in Section 5 of RFC
2252 where incorporated into this document.

The 'supportedExtension' description was clarified. A server need
only list the OBJECT IDENTIFIERs associated with the extended

requests of the extended operations it recognizes.

The 'supportedControl' description was clarified. A server need
only list the OBJECT IDENTIFIERs associated with the request
controls it recognizes.

A.2.3 Section 7 of RFC 2252

Section 7 of RFC 2252 provides definitions of the 'subschema' and
'extensibleObject' object classes. These definitions where
integrated into Section 4.2 and Section 4.3 of this document,
respectively. Section 7 of RFC 2252 also contained the object class
implementation requirement. This was incorporated into Section 7 of
this document.

The specification of 'extensibleObject' was clarified of how it
interacts with precluded attributes.

A.3 Changes to RFC 2256

This document incorporates Sections 5.1, 5.2, 7.1, and 7.2 of RFC
2256.

Section 5.1 of RFC 2256 provided the definition of the 'objectClass'
attribute type. This was integrated into Section 2.4.1 of this
document. The statement "One of the values is either 'top' or
'alias'" was replaced with statement that one of the values is 'top'
as entries belonging to 'alias' also belong to 'top'.

Section 5.2 of RFC 2256 provided the definition of the
'aliasedObjectName' attribute type. This was integrated into
Section 2.6.2 of this document.

Section 7.1 of RFC 2256 provided the definition of the 'top' object
class. This was integrated into Section 2.4.1 of this document.

Section 7.2 of RFC 2256 provided the definition of the 'alias'
object class. This was integrated into Section 2.6.1 of this
document.