[an error occurred while processing this directive] [an error occurred while processing this directive] [an error occurred while processing this directive] [an error occurred while processing this directive] [an error occurred while processing this directive]

Chapter 3. LDAP Schemas, objectClasses and Attributes

This Chapter is not for the faint-hearted. It starts to drill down into the nauseous detail. You can either read it now or go to the Samples section and 'do stuff'. The samples have tons of links back to this chapter to explain specific items in detail.

LDAP and X.500 are feet deep in terminology. Some terminology is important, some is just fluff.

We have created a glossary to jog your memory and introduce terms, either because they are important or because they are frequently used in the literature. We occasionally use terms that we think are more understandable, but will normally then follow them with the standard LDAP term.

Because Schemas, objectClasses and Attributes are so interrelated, we use the highly technical term **stuff** to describe them collectively. You can substitute the infinitely more precise term **thingies** or whatever else you choose.

When you create an entry in a DIT its data contents are contained in **attributes**, which are grouped into **objectclasses**, which are packaged into **schemas**.

The complexity and power of LDAP comes from the fact that there are bucket loads of attributes and bucket loads of objectclasses liberally scattered round in apparently random (and invariably unhelpfully) named schemas. Either you stick to some well-known applications, in which case you can use the well-known objectclasses and attributes, or you can invest some time to learn their language so that you can discover which objectclasses and attributes are truly best for your application - or even create your own.

We are slowly making the standard schemas browsable. If we were smart and had the time, we would modify **htags** and **gtags** to do this. But, since we are neither smart nor do we have the time....

Contents

3.1 LDAP Stuff Overview

Everything in LDAP is hierarchical - so also with **objectclasses** and **attributes**. **Schemas** are important but not terribly interesting, providing the packaging units that **roughly** group together related **objectclasses** and **attributes**.

The important rules regarding each 'thingy' are defined below:

1. **Schemas** are simply packaging units - they are collections of thingies if you need the precise definition - and a short hand method of referencing a lot of stuff (thingies such as ojectclasses and attributes) rather than referencing the thingies individually:

- All **objectclasses** and **attributes** are defined inside schemas (there are some objectclasses and attributes defined as being operational which are embedded in the LDAP server software and do not need external definition, but we will ignore them just now).
- All the schemas which include the **objectclasses** and **attributes** used in any LDAP implementation must be known to the LDAP server (in OpenLDAP under olc (cn=config) either as part of the standard installation or they can be added using this procedure or by the include statement in the slapd.conf configuration file).
- An **attribute** defined in one **schema** can be used by an **objectclass** defined in another **schema** - Pick'n Mix style.

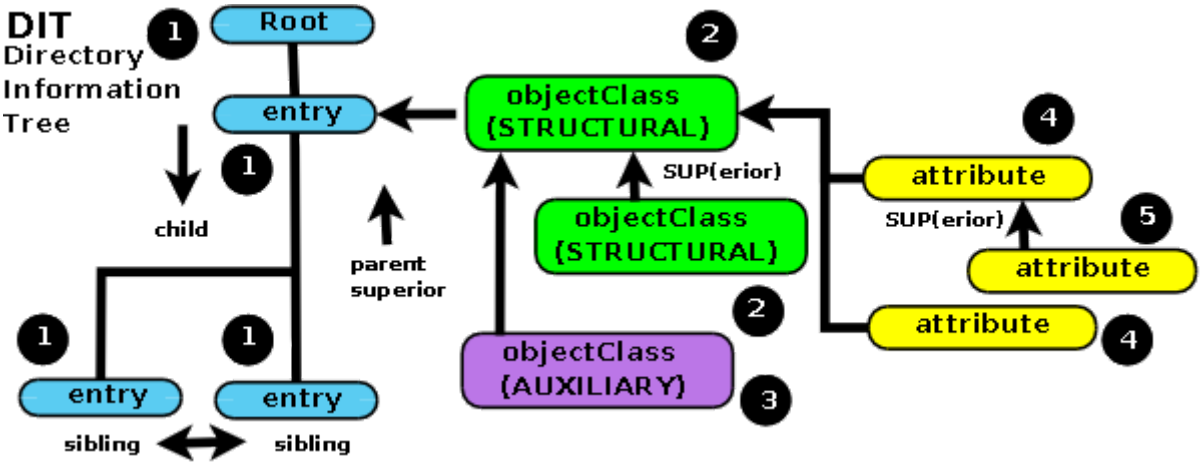
2. **objectClasses** group sets of **attributes**:
   - **objectclasses** are defined inside **schemas**.
   - **objectclasses** may be organised in a hierarchy in which case they inherit all the properties of their parents or SUPerior in the LDAP jargon.
   - **objectclasses** may be STRUCTURAL, in which case they can be used to create entries (data objects), AUXILIARY in which case they may be added into any convenient entry, or ABSTRACT - a non-existent 'thingie'. The most common ABSTRACT objectclass is **top**, which forms the highest level of every objectclass hierarchy, and terminates any hierarchy.
   - If an **objectclass** is part of a hierarchy it must be the same type (STRUCTURAL, AUXILIARY) as any SUPerior **objectClass**. The exception to this rule is if the SUPerior is the **top** ABSTRACT type which as noted is used to terminate any hierarchy.
   - **objectclasses** are the means for including attributes (they are attribute containers in the jargon).
   - **objectclasses** define whether an attribute is mandatory (MUST be present) or optional (MAY be present) within the objectClass.
   - **objectclasses** are defined using ASN.1 notation.
3. **Attributes** typically contain data:
   - Every **attribute** is defined in a **schema**.
   - Every **attribute** is included in one or more **objectclasses**.
   - To use an **attribute** in an entry, its **objectclass** must be included in the entry definition and its **objectclass** must be included in a **schema**. In turn, the schema must be identified to the LDAP server.
   - An **attribute**'s characteristics are defined using ASN.1 notation.
   - An **attribute** can appear once in any instance of its containing **ObjectClass** (SINGLE-VALUE) or can appear more than once in any instance of its containing **ObjectClass** (MULTI-VALUE). MULTI-VALUE is the default. Thus, for example, it is perfectly reasonable to have more that one instance of an email address (attribute mail) but it would be slightly confusing to have more that one instance of a password (attribute userPassword). Not even your mother could sort that out.
   - An **attribute** definition may be part of a hierarchy, in which case it inherits all the properties of its parents. For example, commonName (cn), givenName (gn) and surname (sn) are all children of the **name** attribute. Unlike the **objectClass**, attribute hierarchies are not terminated with a **top** equivalent. In the attribute case it is the absence of a SUPerior definition which indicates, surprisingly, that this is the end of the hierarchy.

- An **attribute** definition includes its type (or SYNTAX), for example, a string or number, and how it behaves in certain conditions, for instance, whether comparison operations are case-sensitive or case-insensitive using what are called matchingRules (more on this later, much later).

4. **entries** group sets of **objectclasses** within a DIT:

   - **entries** must contain one, and only one, STRUCTURAL **objectClass**. A STRUCTURAL **objectClass** may have a SUPerior (may be part of a hierarchy) which is also STRUCTURAL and thus the hierarchy may be viewed as a single STRUCTURAL **objectClass**.
   - **entries** may contain any number of AUXILIARY **objectClasses**.
   - **entries** can have **child** entries which appear below them in the address (naming) hierarchy
   - **entries** can have **parent** entries which appear above them in the address (naming) hierarchy
   - **entries** can have **sibling** entries which appear at the same level as them in the address hierarchy. **sibling** entries share a common **parent** entry.
   - **entries** may be of three types; an **object entry** (the most common entry type) consisting of user data contained in attributes within objectClasses; an **alias entry** having the objectClass alias with the single attribute **aliasedObjectName**; a subentry which is used to store administrative or operational data related (in some way) to its parent entry.

The following diagram illustrates some of these relationships:



Up Arrow

3.2 LDAP Schemas

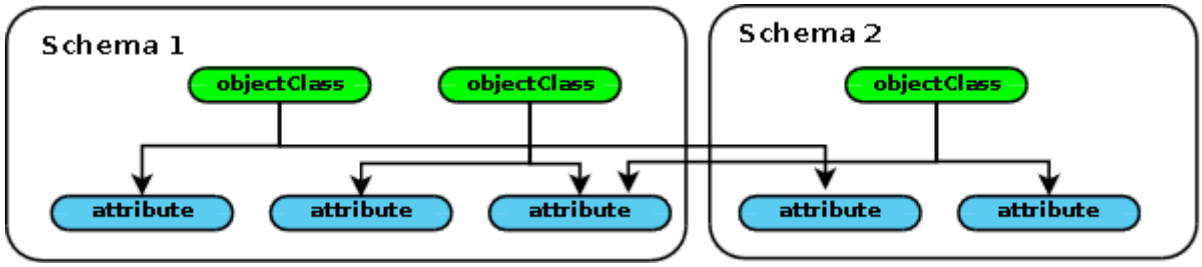
An **LDAP schema** is nothing more than a convenient packaging unit for containing broadly similar objectClasses and attributes.

There may have been a time when a single schema was designed to hold everything required for an LDAP implementation (like a relational database schema) but that is no longer true. You will find useful attributes and objectclasses scattered all over the place - the power of LDAP arguably comes from the ease of creating and using this apparent anarchy.

The rule is: Every attribute or objectclass, including its superior objectclass or attribute, used in an LDAP implementation must be defined in a **schema**, and that schema must be **known** to the LDAP server by a configuration procedure or option. In OpenLDAP OLC (cn=config) the installed schemas are located under cn=schema, cn=config and additional schemas may be installed using this procedure.

If using slapd.conf they use the include statement.

The following diagram illustrates the use of schemas as packaging units:



Up Arrow

3.3 LDAP objectClasses

An **objectClass** is a collection of attributes (or an attribute container) and has the following characteristics:

1. An **objectclass** is defined within a **Schema**
2. An **objectclass** may be a part of an objectclass hierarchy, in which case it inherits all the properties (which include the contained attributes) of its parent(s). For example, inetOrgPerson is the child of organizationalPerson, which is the child of person, which is the child of **top** (the ABSTRACT objectClass which terminates every objectClass hierarchy).
3. An **objectclass** has a globally unique name or identifier
4. An **objectclass**, as well as being an attribute container, is also an attribute and can appear in a search operation.
5. An **objectclass** defines its member attributes and whether these MUST be present (mandatory) or MAY be present (optional) in an entry.
6. One or more **objectclass(es)** must be present in an LDAP entry.
7. One and only one STRUCTURAL **objectclass** must be present in an LDAP entry.
8. Each **objectclass** supported by an LDAP server forms part of a **collection** called **objectclasses** which can be discovered via the subschema.

Defining an objectClass

The formal objectclass definition is defined in RFC 4512 section 4.1.1 and looks like this:

```
ObjectClassDescription = "(" whsp
 numericoid whsp      ; ObjectClass identifier
 [ "NAME" qdescrs ]
 [ "DESC" qdstring ]
 [ "OBSOLETE" whsp ]
 [ "SUP" oids ]       ; Superior ObjectClasses
 [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                      ; default structural
 [ "MUST" oids ]      ; AttributeTypes
 [ "MAY" oids ]       ; AttributeTypes
  extensions
        whsp ")"
```

Ooof! **whsp** means a space (or tab, LF, CR or FF) character, and when they say it should be there, believe them. Rather than try and explain all these parameters individually, let's start with some examples.

An **objectClass** is defined using ASN.1 notation - the following is a simple standard objectclass definition for country taken from the core.schema supplied with OpenLDAP distributions.

```
objectclass ( 2.5.6.2 NAME 'country' DESC 'RFC2256: a country'
  SUP top STRUCTURAL
  MUST c
  MAY ( searchGuide $ description ) )
```

Now let's deconstruct this definition:

objectclass is a keyword indicating this is an objectclass definition - see it's not so complicated.

2.5.6.2 NAME 'country' defines a **globally unique** name for this objectclass and is comprised of two parts: **NAME 'country'** just allows you to refer to this objectclass by some semi-understandable text - in this case the english word **country**. The **globally unique** part is defined by **2.5.6.2** which is called an OID (ObjectIdentifier). The OID 2.5.6.2 was likely the third objectclass ever defined by X.500 (deduced from the fact that 2.5.6 defines the joint itu-iso x.500 object classes, the last 2 is a sequence number within that family of OIDs). It does not matter what organization assigns this number but it must be UNIQUE. Obtaining an enterprise OID, formally a Private Enterprise Number (PEN), that allows you to define your own **attributes** and **objectclasses** is a trivial and zero cost process via IANA. It is a VERY BAD THING™ to re-use existing OIDs.

SUP 'top' indicates that this objectclass has a parent (or SUPerior) objectclass - it is part of a hierarchy. In this case the parent is **top** which is a special ojectclass (ABSTRACT) that terminates (is the highest level) in all objectclasses. An objectclass may have one or more objectclass(es) as Parents.

The keyword **STRUCTURAL** indicates that this objectclass contains attributes and can form an entry in a DIT. There can be only one STRUCTURAL objectClass in an entry but a STRUCTURAL objectclass may be part of a hierarchy (has other STRUCTURAL objectClass(es) as a SUP). (More information about STRUCTURAL hierarchies and Inheritance.) **objectClasses** may also be ABSTRACT which indicates a non-existent objectclass used for convenience. The most common ABSTRACT objectclass is **top** which just terminates an objectclass hierarchy. Finally, an **objectClass** may be AUXILIARY which indicates it contains attributes and may be used with any STRUCTURAL objectclass to form an entry, but cannot alone form an entry in a DIT. All entries need one (and only one) STRUCTURAL objectClass. All entries may have zero or more AUXILIARY objectClass(es).

DESC 'a country' OK, so we picked a lousy example which does not have a meaningful DESC part - but it was a short objectClass. DESC is an optional value that allows a text description of the use or contents of the objectclass. It's meant for human beings to read and has no other use. Here is what **country** should have looked like with a sensible DESC statement included:

```
objectclass ( 2.5.6.2 NAME 'country' SUP top STRUCTURAL
  DESC 'A geographic entity described by a 2 letter ISO 3166 assigned country code'
  MUST c
  MAY ( searchGuide $ description ) )
```

DESC values run the gamut from the non-existent through the pathetic to the misleading with the occasionally useful one thrown in to confuse everyone.

MUST c MUST indicates that the attributes in the following list are mandatory. In this case the attribute **c** (c or countryName) has to be present or an objectClass instance will not be created (it will fail with a nasty error message) and if this is a STRUCTURAL objectClass an entry will not be created (it will fail with an even nastier error message). Single values are written as shown, multiple attributes are enclosed in parentheses and separated with a $ (dollar sign), such as **(attr1 $ attr2 $ attrn)**. If there are no mandatory (MUST) attributes this section is not present.

MAY (searchGuide $ description) MAY indicates that the attributes in the following list are optional and do not need to be present in order to create an instance of the objectClass. Multiple values are written as shown, whereas single attributes do not need the parentheses (see MUST attributes above for singleton syntax). If there are no optional (MAY) attributes this section is not present.

Some more objectClasses

This is how the **top** objectclass is defined:

```
objectclass ( 2.5.6.0 NAME 'top' ABSTRACT
        MUST objectClass )
```

Illustrates the use of the ABSTRACT statement in an objectclass. Since **top** is always the top of a hierarchy clearly it cannot have a **SUP** statement. The OID is also assigned by the X.500 standards group.

Many documents insist that the objectclass **top** is included in LDIF files - it is not always necessary.

This is how the **dcObject** objectclass is defined:

```
objectclass ( 1.3.6.1.4.1.1466.344 NAME 'dcObject'
        DESC 'RFC2247: domain component object'
        SUP top AUXILIARY MUST dc )
```

This example illustrates the use of the AUXILIARY statement. It is not possible to create an entry based on a single AUXILIARY statement. The OID in this example shows the use of a private enterprise OID (ObjectIdentifier). The following fragment shows a fairly typical base DN definition using **dcObject**:

```
dn: dc=example,dc=com
dc: example.com
objectclass: dcObject
objectclass: organization
o: Example, Inc.
```

It is the **objectclass: organization** (a STRUCTURAL objectClass) that creates the entry. **dcObject** piggy-backs on this objectclass.

This is how the **pilotOrganization** objectclass is defined and illustrates that there may be one or more SUPerior (Parent) objectclasses (in this example both **organization** and **organizationalUnit** are parents), in which case the child inherits the properties of ALL its parents (a bit like humans really): (More information about inheritance in LDAP.)

```
objectClasses: ( 0.9.2342.19200300.100.4.20 NAME 'pilotOrganization'
  SUP ( organization $ organizationalUnit ) STRUCTURAL
  MAY buildingName )
```

We have omitted the explanation of the OBSOLETE term. If it is present in the objectClass definition, it means the objectclass should not be used (duh!).

Up Arrow

3.4 LDAP Attributes

Attributes typically contain data and have the following characteristics:

1. Every **attribute** is defined in a **schema**.
2. Every **attribute** is included in one or more **objectclasses**.
3. An **objectclass** is also an **attribute** and can be used in searches.
4. To use an **attribute** in an entry, its **objectclass** must be included in the entry definition, and its **objectclass** must be included in a **schema**. The schema, in turn, must must be identified to the LDAP server.
5. An **attribute**'s characteristics are defined using ASN.1 notation.
6. An **attribute** can appear once in any instance of its containing **ObjectClass** (SINGLE-VALUE) or can appear more than once in any instance of its containing **ObjectClass** (MULTI-VALUE). MULTI-

VALUE is the default.

7. An **attribute** definition may be part of a hierarchy, in which case it inherits all the properties of its parents. For example, commonName (cn), givenName (gn), surname (sn) are all children of the **name** attribute.

8. An **attribute** definition includes its type (or SYNTAX), for example, a string or number, and how it behaves in certain conditions; for instance, whether comparison operations are case-sensitive or case-insensitive.

9. An **attribute** supported by an LDAP server forms part of a **collection** called **attributetypes** which can be interrogated via the subschema.

Defining an Attribute

The formal attribute definition is defined in RFC 4512 section 4.1.2 and looks like this:

```
AttributeTypeDescription = "(" whsp
   numericoid whsp     ; AttributeType identifier
 [ "NAME" qdescrs ]            ; name used in AttributeType
 [ "DESC" qdstring ]           ; description
 [ "OBSOLETE" whsp ]
 [ "SUP" oid ]                 ; derived from this other
                               ; AttributeType
 [ "EQUALITY" woid             ; Matching Rule name
 [ "ORDERING" woid             ; Matching Rule name
 [ "SUBSTR" woid ]             ; Matching Rule name
 [ "SYNTAX" whsp noidlen whsp ] ; Syntax OID
 [ "SINGLE-VALUE" whsp ]       ; default multi-valued
 [ "COLLECTIVE" whsp ]         ; default not collective
 [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
 [ X-ORDERED whsp type ]       ; non-standard - default not X-ORDERED
 [ "USAGE" whsp AttributeUsage ]; default userApplications
 extensions
 whsp ")"
```

whsp means a space (or TAB, LF, CR, FF) character and must be present. Rather than explain each bit of gobbledegook let's again start with some examples:

An **attribute** is defined using ASN.1 notation - the following is a simple standard attribute definition for commonName (cn) taken from the core.schema supplied with OpenLDAP distributions.

```
attributetype ( 2.5.4.3 NAME ( 'cn' 'commonName' ) SUP name )
```

Now lets deconstruct this definition:

attributetype indicates this defines an attribute - wow, we got this stuff cold dude.

2.5.4.3 NAME ('cn' 'commonName') defines a **globally unique** name for this attribute and is comprised of two values in parenthesis: **NAME ('cn' 'commonName')**. This allows you to refer to this attribute by some semi-understandable text - in this case either the english word **commonName** OR the shortform (alias) **cn**. In principle there are no limits to the number of shortforms or aliases you can have as long as they are unique. In this multiple entry form the names are enclosed in parentheses and separated by a space. Since **cn** appears first, it is called the primary name, which is very important when it comes to indexing entries for search optimization.

The **globally unique** part is defined by **2.5.4.3** which is an OID (ObjectIdentifier). The OID 2.5.4.3 was possibly the fourth attribute ever defined by X.500 (2.5.4 is the joint itu-iso x.500 attribute types, the last 3 is a sequence number within that family of OIDs). It does not matter what organization assigns this number but it must be UNIQUE. Obtaining an enterprise OID that allows you to define your own **attributes** and **objectclasses** is a trivial process via IANA. It is an EXTREMELY BAD THING™ to re-use existing OIDs.

SUP 'name' indicates that this attribute has a parent (or SUPerior) attribute - it is part of a hierarchy. In this case the parent is **name** which we will now look at in detail since, if you recall, the child always inherits the properties of the parent (or SUPerior) attribute (and itself may have additional properties). The SUP entry can use either a 'name' or an OID. The definition SUP 'name' and SUP 2.5.4.41 mean exactly the same - except to the poor human - and can be interchanged at will.

This is the attribute definition of name which is a much more serious definition and the SUPerior (parent) attribute of **cn** above:

```
attributetype ( 2.5.4.41 NAME 'name'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )
```

Now for some more serious deconstruction:

attributetype indicates this defines an attribute - same as before.

2.5.4.41 NAME 'name' defines the **globally unique** name for this attribute and as before is comprised of two parts: **NAME 'name'** just allows reference to this attribute by some semi-understandable text and the OID **2.5.4.41** indicates it was defined by the X.500 standards group. The format used, because there is only a single name value, does not need enclosing parenthesis as in the **commonName** example above.

EQUALITY caseIgnoreMatch indicates how this (and any child attributes) will behave when used in a search filter, for instance, **(cn=jimbob)** (**cn** is a child of **name**) and no **wildcards** exist in the search. In this case it defines the match to be case-insensitive. **caseIgnoreMatch** is a matchingRule and is defined in the subschema.

SUBSTR caseIgnoreSubstringsMatch indicates how this (and any child attributes) will behave when used in a search filter which uses a substring, for example, **(cn=jim*)** (**cn** is a child of **name**) and contains one or more **wildcards**. In this case it defines that the match is case-insensitive. **caseIgnoreSubstringMatch** is again a matchingRule and is defined in the subschema.

SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} is an OID which defines the data type and what rules (data validation) are applied to the data. The full list is in RFC 2252 section 4.3.2. In this case the OID defines it to be a Directory String type which is defined in RFC 2252 section 6.10 to be in the UTF-8 form of the ISO 10646 character set. The value **{32768}** indicates the maximum length of the string and is optional. (More info on LDAP Data Types.)

Other Characteristics

SINGLE-VALUE Omission of this entry means that it is multi-valued, that is, it can appear more than once in an **objectclass**. If the attribute can only accept single values (can only appear once in any objectClass) it must be explicitly defined as in the definition of **dc** below.

```
attributetype ( 0.9.2342.19200300.100.1.25
  NAME ( 'dc' 'domainComponent' )
  DESC 'RFC1274/2247: domain component'
  EQUALITY caseIgnoreIA5Match
  SUBSTR caseIgnoreIA5SubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26
  SINGLE-VALUE )
```

USAGE 'AttributeUsage' the default is **userApplication** (attribute is returned with the * value in a search string) but the value dSAOperation or directoryOperation defines the attribute to be operational (returned with the + value in a search string).

ORDERING 'matchingrule' is rarely defined and is used to define the collation match - the lexicographic sorting order (allowing searches of <= and >=) as in the attribute definition below:

```
attributetype ( 2.5.4.46 NAME 'dnQualifier'
        EQUALITY caseIgnoreMatch
        ORDERING caseIgnoreOrderingMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.44 )
```

COLLECTIVE Defined by RFC 3671. COLLECTIVE is only valid with USAGE 'userApplication' (the default) and if present in an attribute definition allows all instances of the attribute's SUPerior to share the same value, for example a **telephoneNumber** attribute. By convention the name allocated to the collective attribute will be the SUPerior attribute name preceded by **c-**, thus in the COLLECTIVE attribute with a SUPerior of **telephoneNumber** is **c-telephoneNumber**. An example definition of a COLLECTIVE attribute is shown:

```
( 2.5.4.20.1 NAME 'c-TelephoneNumber'
     SUP telephoneNumber COLLECTIVE )
```

Collective attributes appear as subentries of the objectClass **collectiveAttributeSubentry**. By modifying the COLLECTIVE attribute(s) (via their subentries) all instances of the SUPerior attributes can be changed.

NO-USER-MODIFICATION If present this indicates the attribute may not be modified by the user irrespective of any assigned permissions in ACL/Security statement. It is typically only present when the **USAGE** is dsaOperation or directoryOperation. If not present then the attribute may be modified by the user (subject to any ACL/Security parameters).

X- Any LDAP object may define additional properties over and above those defined by the relevant standards (though clearly they must be recognized by one or more LDAP server implementations to be useful). All such property names must begin with X- (see X-ORDERED below for an example recognized by OpenLDAP) and must have a single quoted parameter, for example, X-MY-PROPERTY 'TRUE'.

X-ORDERED:

These notes can be mercifully skipped for most readers and are provided only for those working in the bowels of OpenLDAP or who are trying to figure out what those {} mean when working with OpenLDAP's on-line configuration(OLC cn=config) system.

1. X-ORDERED 'type' is a non-standard element (currently, and incompletely, defined by draft-chu-ldap-xordered-00) and is used extensively in OpenLDAP's OLC (cn=config) feature. The use of this element in an attribute definition enables the creation of ordered sets.

2. **type** may be either 'VALUES' or 'SIBLINGS'.

3. 'VALUES' is used only with MULTI-VALUE attributes (denoted by the absence of SINGLE-VALUE in the attribute definition) and indicates that each attribute will have an order element of the form {x} prepended to its value (which then becomes part of the attribute's value) where x is a numeric value starting from 0. This allows multi-valued attributes to be addressed explicitly (for modify or delete operations) and for new attributes to be inserted in order where it is important or essential, for example, when using ACLs/ACPs in the **olcAccess** attribute.

4. 'SIBLINGS' may only be used in SINGLE-VALUE attribute definitions. Where an attribute with this value is present in an entry it indicates that its immediate child entries (one level only) must have an ordered value {x} in their DN. The ordered value prefix may be added explicitly when the child entry is added or if not present it will be allocated a sequentially ordered prefix, starting from 0, when the child entry is added.

Up Arrow

3.5 Matching Rules

Matching rules are part of what is called the operational characteristics of the LDAP server.

matchingrules define the methods of comparison available in the LDAP server:

1. **matchingrules** are typically built-in to the LDAP server and do not need to be defined explicitly.
2. A **matchingrule** forms part of a **collection** called **matchingrules** which can be discovered via the subschema.
3. A **matchingrule** is defined for each **attribute** using the EQUALITY, SUBSTR and ORDERING properties as required - only those properties required are defined. So, if the attribute will not support the use of a **wildcard** in a search there will be no SUBSTR property defined.

3.5.1 Defining matchingRule

Most **matchingrules** are built-in and you almost never need to define them, but like everything in LDAP it has a defining syntax. The following is an example of a matchingrule definition using **caseIgnoreMatch**:

```
matchingRule ( 2.5.13.2 NAME 'caseIgnoreMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The deconstruction shows the following:

matchingrule indicates this is the start of a matchingrule definition.

2.5.13.2 NAME 'caseIgnoreMatch' defines the **globally unique** name for this matching rule and as always is comprised of two parts: **NAME 'caseIgnoreMatch'** allows reference to this matchingrule using some semi-understandable text and the OID **2.5.13.2** indicates the matching rule was defined by the X.500 standards group. Rule description:

"The Case Ignore Match rule compares for equality a presented string with an attribute value of type PrintableString, NumericString, TeletexString, BMPString, UniversalString or DirectoryString without regard for case (upper or lower) of the strings (e.g., "Dundee" and "DUNDEE" match).

The rule returns TRUE if the strings are the same length and corresponding characters are identical except possibly with regard to case.

SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 defines that this matchingrule operates on the type(s) defined - in this case a DirectoryString (a UTF-8 format string).

OpenLDAP built-in matchingRules

This list below can be found for OpenLDAP by interrogating the **subschema** using a command like:

```
ldapsearch -H ldap://ldap.example.com -x -s base -b "cn=subschema"
  "(objectclass=*)" matchingrules
# matchingrules may be changed to
# attributetypes objectclasses etc., etc.
```

The above command should be on a single line - it is split for HTML formatting reasons only. Replace ldap.example.com with the host name of your LDAP server. If the server is running locally you can omit the -H argument (defaults to **localhost**).

Alternatively use any decent LDAP browser with a base DN of "cn=subschema"

The above command will return something like this list:

```
# Subschema
dn: cn=Subschema
matchingRules: ( 2.5.13.0 NAME 'objectIdentifierMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
matchingRules: ( 2.5.13.1 NAME 'distinguishedNameMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
matchingRules: ( 2.5.13.2 NAME 'caseIgnoreMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

```
matchingRules: ( 2.5.13.3 NAME 'caseIgnoreOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
matchingRules: ( 2.5.13.4 NAME 'caseIgnoreSubstringsMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
matchingRules: ( 2.5.13.5 NAME 'caseExactMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
matchingRules: ( 2.5.13.6 NAME 'caseExactOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
matchingRules: ( 2.5.13.7 NAME 'caseExactSubstringsMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
matchingRules: ( 2.5.13.8 NAME 'numericStringMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.36 )
matchingRules: ( 2.5.13.10 NAME 'numericStringSubstringsMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
matchingRules: ( 2.5.13.13 NAME 'booleanMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
matchingRules: ( 2.5.13.14 NAME 'integerMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
matchingRules: ( 2.5.13.15 NAME 'integerOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
matchingRules: ( 2.5.13.16 NAME 'bitStringMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )
matchingRules: ( 2.5.13.17 NAME 'octetStringMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )
matchingRules: ( 2.5.13.18 NAME 'octetStringOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )
matchingRules: ( 2.5.13.20 NAME 'telephoneNumberMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50 )
matchingRules: ( 2.5.13.21 NAME 'telephoneNumberSubstringsMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.58 )
matchingRules: ( 2.5.13.23 NAME 'uniqueMemberMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.34 )
matchingRules: ( 2.5.13.27 NAME 'generalizedTimeMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
matchingRules: ( 2.5.13.28 NAME 'generalizedTimeOrderingMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
matchingRules: ( 2.5.13.29 NAME 'integerFirstComponentMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
matchingRules: ( 2.5.13.30 NAME 'objectIdentifierFirstComponentMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
matchingRules: ( 2.5.13.34 NAME 'certificateExactMatch'
  SYNTAX 1.2.826.0.1.3344810.7.1 )
matchingRules: ( 1.3.6.1.4.1.1466.109.114.1 NAME 'caseExactIA5Match'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
matchingRules: ( 1.3.6.1.4.1.1466.109.114.2 NAME 'caseIgnoreIA5Match'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
matchingRules: ( 1.3.6.1.4.1.1466.109.114.3 NAME 'caseIgnoreIA5SubstringsMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
matchingRules: ( 1.3.6.1.4.1.4203.1.2.1 NAME 'caseExactIA5SubstringsMatch'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
matchingRules: ( 1.2.840.113556.1.4.803 NAME 'integerBitAndMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
matchingRules: ( 1.2.840.113556.1.4.804 NAME 'integerBitOrMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
```

You can find what the **OIDs** are and therefore the exact english description of the matchingRule using this great site or alternatively this terrific site.

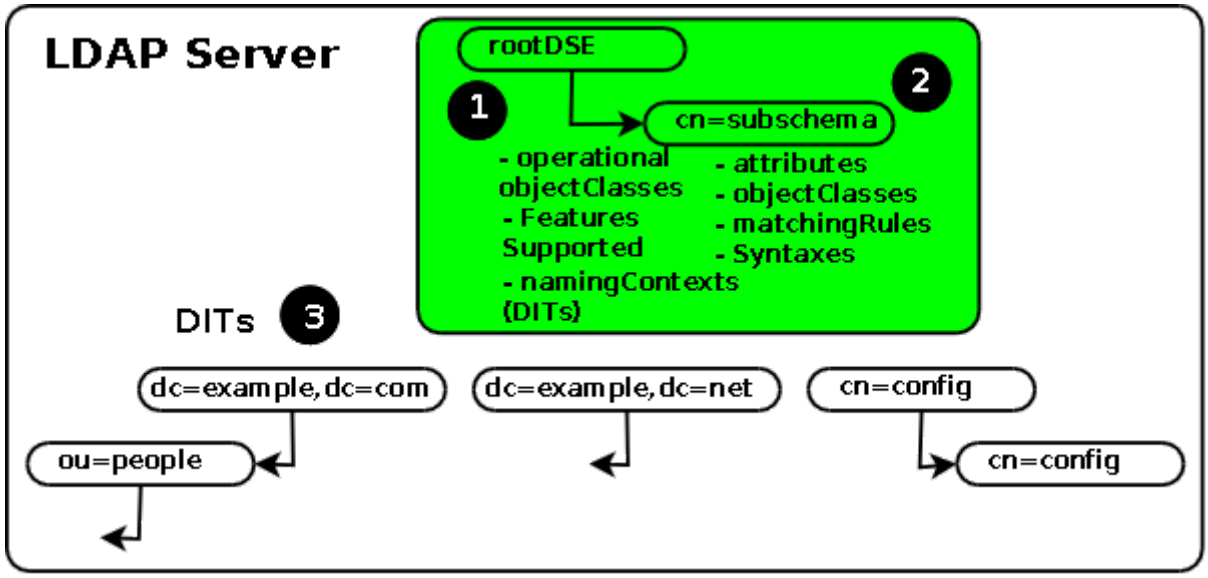
Up Arrow

3.6 LDAP Operational Attributes and Objects

There are a bunch of attributes and objectclasses built into the LDAP server that govern how it functions. These attributes and object classes are typically referred to as **operational**.

These **operational thingies** all live under the rootDSE and are not visible in normal operations.

The relationship between the DIT(s) and its entries and the RootDSE (Root DSA (Directory System Agent) - Specific Entry) and its objects is shown below:



The rootDSE can be inspected using either a suitable LDAP browser (instructions for LDAPBrowser/Editor) with an empty base DN or the following command:

```
ldapsearch -H ldap://ldap.example.com -x -s base -b "" +
# note the + returns operational attributes
```

This should return something similar to that shown below (from OpenLDAP 2.4.8) - the values in parentheses are added explanations and are not returned by the server:

```
dn:
structuralObjectClass: OpenLDAProotDSE
configContext: cn=config
namingContexts: dc=example,dc=com
namingContexts: dc=example,dc=net
monitorContext: cn=Monitor
supportedControl: 1.3.6.1.4.1.4203.1.9.1.1 (Contentsync RFC 4530)
supportedControl: 2.16.840.1.113730.3.4.18 (ProxiedAuthv2 RFC 4370)
supportedControl: 2.16.840.1.113730.3.4.2 (ManageDSAIT RFC3377)
supportedControl: 1.3.6.1.4.1.4203.1.10.1 (SubEntries RFC3673)
supportedControl: 1.2.840.113556.1.4.319 (pagedResults RFC2696)
supportedControl: 1.2.826.0.1.3344810.2.3 (MatchedValues RFC3876)
supportedControl: 1.3.6.1.1.13.2 (Post Read RFC4527)
supportedControl: 1.3.6.1.1.13.1 (Pre-Read RFC4527))
supportedControl: 1.3.6.1.1.12 (Assertion RFC4528)
supportedExtension: 1.3.6.1.4.1.4203.1.11.1 (ModifyPassword RFC3088)
supportedExtension: 1.3.6.1.4.1.4203.1.11.3 (WhoAmI RFC4532)
supportedExtension: 1.3.6.1.1.8 (Cancel RFC3909)
supportedFeatures: 1.3.6.1.1.14 (Modify-Increment RFC4525)
supportedFeatures: 1.3.6.1.4.1.4203.1.5.1 (OperationalAttrs RFC3674)
supportedFeatures: 1.3.6.1.4.1.4203.1.5.2 (ObjectClassAttrs RFC4529)
supportedFeatures: 1.3.6.1.4.1.4203.1.5.3 (TrueFalse RFC4526)
supportedFeatures: 1.3.6.1.4.1.4203.1.5.4 (LanguageTag RFC3866)
supportedFeatures: 1.3.6.1.4.1.4203.1.5.5 (LanguageRange RFC3866)
supportedLDAPVersion: 3
supportedSASLMechanisms: NTLM
supportedSASLMechanisms: GSSAPI
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: CRAM-MD5
entryDN:
subschemaSubentry: cn=Subschema
```

An explanation of each **supportedExtension** can be found using this great site or alternatively this terrific site. The above listing shows this LDAP server supports two DITs - shown as **namingContexts** - which were configured using this process as well as the **configContext: cn=config** indicating use of OLC (cn=config).

It is possible to add extensions to any LDAP server using olcRootDSE when using OLC (cn=config) or the rootDSE directive when using slapd.conf.

Up Arrow

subSchema subentry objects

The interesting **stuff** is under **subschema** subentry which you can inspect with a decent LDAP browser (instructions for LDAPBrowser/Editor) using a base DN of "cn=subschema" or use the following command:

```
ldapsearch -H ldap://ldap.mydomain.com -x -s base -b "cn=subschema" objectclasses
# the list of attributes that may be listed are
# matchingruleuse ldapsyntaxes matchingrules attributetypes
# the above entries are collections
# createtimestamp modifytimestamp
# if you use + alone you will get a huge list of
# everything the LDAP server knows about.
```

The above command will generate the following list:

Note: this LDAP server included the cosine.schema, core.schema, nis.schema, inetorgperson.schema.

```
# Subschema
dn: cn=Subschema
objectClasses: ( 2.5.6.0 NAME 'top'
  DESC 'top of the superclass chain'
  ABSTRACT
  MUST objectClass )
objectClasses: ( 1.3.6.1.4.1.1466.101.120.111 NAME 'extensibleObject'
  DESC 'RFC2252: extensible object' SUP top AUXILIARY )
objectClasses: ( 2.5.6.1 NAME 'alias'
  DESC 'RFC2256: an alias' SUP top STRUCTURAL
  MUST aliasedObjectName )
objectClasses: ( 2.16.840.1.113730.3.2.6 NAME 'referral'
  DESC 'namedref: named subordinate referral' SUP top STRUCTURAL
  MUST ref )
objectClasses: ( 1.3.6.1.4.1.4203.1.4.1 NAME ( 'OpenLDAProotDSE' 'LDAProotDSE')
  DESC 'OpenLDAP Root DSE object' SUP top STRUCTURAL
  MAY cn )
objectClasses: ( 2.5.17.0 NAME 'subentry' SUP top STRUCTURAL
  MUST ( cn $ subtreeSpecification ) )
objectClasses: ( 2.5.20.1 NAME 'subschema'
  DESC 'RFC2252: controlling subschema (sub)entry' AUXILIARY
  MAY ( dITStructureRules $ nameForms $ ditContentRules
  $ objectClasses $ attributeTypes $ matchingRules $ matchingRuleUse ) )
objectClasses: ( 1.3.6.1.4.1.4203.666.3.2 NAME 'monitor'
  DESC 'OpenLDAP system monitoring' STRUCTURAL MUST cn )
objectClasses: ( 2.5.6.2 NAME 'country' DESC 'RFC2256: a country'
  SUP top STRUCTURAL MUST c MAY ( searchGuide $ description ) )
objectClasses: ( 2.5.6.3 NAME 'locality'
  DESC 'RFC2256: a locality' SUP top STRUCTURAL
  MAY ( street $ seeAlso $ searchGuide $ st $ l $ description ) )
objectClasses: ( 2.5.6.4 NAME 'organization'
  DESC 'RFC2256: an organization' SUP top STRUCTURAL
  MUST o
  MAY ( userPassword $ searchGuide $ seeAlso $ businessCategory
   $ x121Address $ registeredAddress $ destinationIndicator
   $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier
   $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber
   $ street $ postOfficeBox $ postalCode $ postalAddress
   $ physicalDeliveryOfficeName $ st $ l $ description ) )
objectClasses: ( 2.5.6.5 NAME 'organizationalUnit'
  DESC 'RFC2256: an organizational unit' SUP top STRUCTURAL
  MUST ou
  MAY ( userPassword $ searchGuide $ seeAlso $ businessCategory
   $ x121Address $ registeredAddress $ destinationIndicator
   $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier
```

```
    $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber
    $ street $ postOfficeBox $ postalCode $ postalAddress
    $ physicalDeliveryOfficeName $ st $ l $ description ) )
  objectClasses: ( 2.5.6.6 NAME 'person'
    DESC 'RFC2256: a person' SUP top STRUCTURAL
    MUST ( sn $ cn )
    MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
  objectClasses: ( 2.5.6.7 NAME 'organizationalPerson'
    DESC 'RFC2256: an organizational person' SUP person STRUCTURAL
    MAY ( title $ x121Address $ registeredAddress $ destinationIndicator
     $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier
     $ telephoneNumber $ internationaliSDNNumber
     $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode
     $ postalAddress $ physicalDeliveryOfficeName $ ou $ st $ l ) )
  objectClasses: ( 2.5.6.8 NAME 'organizationalRole'
    DESC 'RFC2256: an organizational role' SUP top STRUCTURAL
    MUST cn
    MAY ( x121Address $ registeredAddress $ destinationIndicator
     $ preferredDeliveryMethod $ telexNumber
     $ teletexTerminalIdentifier $ telephoneNumber
     $ internationaliSDNNumber $ facsimileTelephoneNumber
     $ seeAlso $ roleOccupant $ preferredDeliveryMethod $ street
     $ postOfficeBox $ postalCode $ postalAddress
     $ physicalDeliveryOfficeName $ ou $ st $ l $ description ) )
  objectClasses: ( 2.5.6.9 NAME 'groupOfNames'
    DESC 'RFC2256: a group of names (DNs)' SUP top STRUCTURAL
    MUST ( member $ cn )
    MAY ( businessCategory $ seeAlso $ owner $ ou $ o $ description ) )
  objectClasses: ( 2.5.6.10 NAME 'residentialPerson'
    DESC 'RFC2256: an residential person' SUP person STRUCTURAL
    MUST l
    MAY ( businessCategory $ x121Address $ registeredAddress
     $ destinationIndicator $ preferredDeliveryMethod $ telexNumber
     $ teletexTerminalIdentifier $ telephoneNumber
     $ internationaliSDNNumber $ facsimileTelephoneNumber
     $ preferredDeliveryMethod $ street $ postOfficeBox
     $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l ) )
  objectClasses: ( 2.5.6.11 NAME 'applicationProcess'
    DESC 'RFC2256: an application process' SUP top STRUCTURAL
    MUST cn
    MAY ( seeAlso $ ou $ l $ description ) )
  objectClasses: ( 2.5.6.12 NAME 'applicationEntity'
    DESC 'RFC2256: an application entity' SUP top STRUCTURAL
    MUST ( presentationAddress $ cn )
    MAY ( supportedApplicationContext $ seeAlso $ ou $ o $ l $ description ) )
  objectClasses: ( 2.5.6.13 NAME 'dSA'
    DESC 'RFC2256: a directory system agent (a server)'
    SUP applicationEntity STRUCTURAL
    MAY knowledgeInformation )
  objectClasses: ( 2.5.6.14 NAME 'device'
    DESC 'RFC2256: a device' SUP top STRUCTURAL
    MUST cn
    MAY ( serialNumber $ seeAlso $ owner $ ou $ o $ l $ description) )
  objectClasses: ( 2.5.6.15 NAME 'strongAuthenticationUser'
    DESC 'RFC2256: a strong authentication user' SUP top AUXILIARY
    MUST userCertificate )
  objectClasses: ( 2.5.6.16 NAME 'certificationAuthority'
    DESC 'RFC2256: a certificate authority' SUP top AUXILIARY
    MUST ( authorityRevocationList $ certificateRevocationList
      $ cACertificate ) MAY crossCertificatePair )
  objectClasses: ( 2.5.6.17 NAME 'groupOfUniqueNames'
    DESC 'RFC2256: a group of unique names (DN and Unique Identifier)'
    SUP top STRUCTURAL
    MUST ( uniqueMember $ cn )
    MAY ( businessCategory $ seeAlso $ owner $ ou $ o $ description ) )
  objectClasses: ( 2.5.6.18 NAME 'userSecurityInformation'
    DESC 'RFC2256: a user security information' SUP top AUXILIARY
    MAY supportedAlgorithms )
  objectClasses: ( 2.5.6.16.2 NAME 'certificationAuthority-V2'
```

```
    SUP certificationAuthority AUXILIARY
    MAY deltaRevocationList )
  objectClasses: ( 2.5.6.19 NAME 'cRLDistributionPoint'
    SUP top STRUCTURAL
    MUST cn
    MAY ( certificateRevocationList $ authorityRevocationList
     $ deltaRevocationList ) )
  objectClasses: ( 2.5.6.20 NAME 'dmd' SUP top STRUCTURAL
    MUST dmdName
    MAY ( userPassword $ searchGuide $ seeAlso $ businessCategory
     $ x121Address $ registeredAddress $ destinationIndicator
     $ preferredDeliveryMethod $ telexNumber
     $ teletexTerminalIdentifier $ telephoneNumber
     $ internationaliSDNNumber $ facsimileTelephoneNumber $ street
     $ postOfficeBox $ postalCode $ postalAddress
     $ physicalDeliveryOfficeName $ st $ l $ description ) )
  objectClasses: ( 2.5.6.21 NAME 'pkiUser'
    DESC 'RFC2587: a PKI user' SUP top AUXILIARY
    MAY userCertificate )
  objectClasses: ( 2.5.6.22 NAME 'pkiCA'
    DESC 'RFC2587: PKI certificate authority' SUP top AUXILIARY
    MAY ( authorityRevocationList $ certificateRevocationList
     $ cACertificate $ crossCertificatePair ) )
  objectClasses: ( 2.5.6.23 NAME 'deltaCRL'
    DESC 'RFC2587: PKI user' SUP top AUXILIARY
    MAY deltaRevocationList )
  objectClasses: ( 1.3.6.1.4.1.250.3.15 NAME 'labeledURIObject'
    DESC 'RFC2079: object that contains the URI attribute type'
    SUP top AUXILIARY MAY labeledURI )
  objectClasses: ( 0.9.2342.19200300.100.4.19 NAME 'simpleSecurityObject'
    DESC 'RFC1274: simple security object' SUP top AUXILIARY
    MUST userPassword )
  objectClasses: ( 1.3.6.1.4.1.1466.344 NAME 'dcObject'
    DESC 'RFC2247: domain component object' SUP top AUXILIARY
    MUST dc )
  objectClasses: ( 1.3.6.1.1.3.1 NAME 'uidObject'
    DESC 'RFC2377: uid object' SUP top AUXILIARY
    MUST uid )
  objectClasses: ( 0.9.2342.19200300.100.4.4 NAME ( 'pilotPerson' 'newPilotPerson' )
    SUP person STRUCTURAL
    MAY ( userid $ textEncodedORAddress
     $ rfc822Mailbox $ favouriteDrink $ roomNumber $ userClass
     $ homeTelephoneNumber $ homePostalAddress $ secretary
     $ personalTitle $ preferredDeliveryMethod $ businessCategory
     $ janetMailbox $ otherMailbox $ mobileTelephoneNumber
     $ pagerTelephoneNumber $ organizationalStatus $ mailPreferenceOption
     $ personalSignature ) )
  objectClasses: ( 0.9.2342.19200300.100.4.5 NAME 'account'
    SUP top STRUCTURAL
    MUST userid
    MAY ( description $ seeAlso $ localityName $ organizationName
    $ organizationalUnitName $ host ) )
  objectClasses: ( 0.9.2342.19200300.100.4.6 NAME 'document'
    SUP top STRUCTURAL
    MUST documentIdentifier
    MAY ( commonName $ description $ seeAlso $ localityName
     $ organizationName $ organizationalUnitName $ documentTitle
     $ documentVersion $ documentAuthor $ documentLocation
     $ documentPublisher ) )
  objectClasses: ( 0.9.2342.19200300.100.4.7 NAME 'room'
    SUP top STRUCTURAL
    MUST commonName
    MAY ( roomNumber $ description $ seeAlso $ telephoneNumber ) )
  objectClasses: ( 0.9.2342.19200300.100.4.9 NAME 'documentSeries'
    SUP top STRUCTURAL
    MUST commonName
    MAY ( description $ seeAlso $ telephonenumber $ localityName
     $ organizationName $ organizationalUnitName ) )
  objectClasses: ( 0.9.2342.19200300.100.4.13 NAME 'domain'
```

```
    SUP top STRUCTURAL
    MUST domainComponent
    MAY ( associatedName $ organizationName $ description
     $ businessCategory $ seeAlso $ searchGuide $ userPassword
     $ localityName $ stateOrProvinceName $ streetAddress
     $ physicalDeliveryOfficeName $ postalAddress $ postalCode
     $ postOfficeBox $ streetAddress $ facsimileTelephoneNumber
     $ internationalISDNNumber $ telephoneNumber
     $ teletexTerminalIdentifier $ telexNumber
     $ preferredDeliveryMethod $ destinationIndicator
     $ registeredAddress $ x121Address ) )
  objectClasses: ( 0.9.2342.19200300.100.4.14 NAME 'RFC822localPart'
    SUP domain STRUCTURAL
    MAY ( commonName $ surname $ description $ seeAlso
     $ telephoneNumber $ physicalDeliveryOfficeName
     $ postalAddress $ postalCode $ postOfficeBox $ streetAddress
     $ facsimileTelephoneNumber $ internationalISDNNumber
     $ telephoneNumber $ teletexTerminalIdentifier
     $ telexNumber $ preferredDeliveryMethod
     $ destinationIndicator $ registeredAddress $ x121Address ) )
  objectClasses: ( 0.9.2342.19200300.100.4.15 NAME 'dNSDomain'
    SUP domain STRUCTURAL
    MAY ( ARecord $ MDRecord $ MXRecord $ NSRecord $ SOARecord
     $ CNAMERecord) )
  objectClasses: ( 0.9.2342.19200300.100.4.17 NAME 'domainRelatedObject'
    DESC 'RFC1274: an object related to an domain'
    SUP top AUXILIARY
    MUST associatedDomain )
  objectClasses: ( 0.9.2342.19200300.100.4.18 NAME 'friendlyCountry'
    SUP country STRUCTURAL
    MUST friendlyCountryName )
  objectClasses: ( 0.9.2342.19200300.100.4.20 NAME 'pilotOrganization'
    SUP ( organization $ organizationalUnit ) STRUCTURAL
    MAY buildingName )
  objectClasses: ( 0.9.2342.19200300.100.4.21 NAME 'pilotDSA'
    SUP dsa STRUCTURAL
    MAY dSAQuality )
  objectClasses: ( 0.9.2342.19200300.100.4.22 NAME 'qualityLabelledData'
    SUP top AUXILIARY
    MUST dsaQuality MAY ( subtreeMinimumQuality $ subtreeMaximumQuality ) )
  objectClasses: ( 2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson'
    DESC 'RFC2798: Internet Organizational Person'
    SUP organizationalPerson STRUCTURAL
    MAY ( audio $ businessCategory $ carLicense $ departmentNumber
     $ displayName $ employeeNumber $ employeeType $ givenName
     $ homePhone $ homePostalAddress $ initials $ jpegPhoto
     $ labeledURI $ mail $ manager $ mobile $ o $ pager
     $ photo $ roomNumber $ secretary $ uid $ userCertificate
     $ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate
     $ userPKCS12 ) )
  objectClasses: ( 1.3.6.1.1.1.2.0 NAME 'posixAccount'
    DESC 'Abstraction of an account with POSIX attributes'
    SUP top AUXILIARY
    MUST ( cn $ uid $ uidNumber $ gidNumber $ homeDirectory )
    MAY ( userPassword $ loginShell $ gecos $ description ) )
  objectClasses: ( 1.3.6.1.1.1.2.1 NAME 'shadowAccount'
    DESC 'Additional attributes for shadow passwords'
    SUP top AUXILIARY MUST uid MAY ( userPassword $ shadowLastChange
    $ shadowMin $ shadowMax $ shadowWarning $ shadowInactive
    $ shadowExpire $ shadowFlag $ description ) )
  objectClasses: ( 1.3.6.1.1.1.2.2 NAME 'posixGroup'
    DESC 'Abstraction of a group of accounts'
    SUP top STRUCTURAL
    MUST ( cn $ gidNumber )
    MAY ( userPassword $ memberUid $ description ) )
  objectClasses: ( 1.3.6.1.1.1.2.3 NAME 'ipService'
    DESC 'Abstraction an Internet Protocol service'
    SUP top STRUCTURAL
    MUST ( cn $ ipServicePort $ ipServiceProtocol )
```

```
    MAY description )
  objectClasses: ( 1.3.6.1.1.1.2.4 NAME 'ipProtocol'
    DESC 'Abstraction of an IP protocol' SUP top STRUCTURAL
    MUST ( cn $ ipProtocolNumber $ description )
    MAY description )
  objectClasses: ( 1.3.6.1.1.1.2.5 NAME 'oncRpc'
    DESC 'Abstraction of an ONC/RPC binding'
    SUP top STRUCTURAL
    MUST ( cn $ oncRpcNumber $ description )
    MAY description )
  objectClasses: ( 1.3.6.1.1.1.2.6 NAME 'ipHost'
    DESC 'Abstraction of a host, an IP device'
    SUP top AUXILIARY
    MUST ( cn $ ipHostNumber )
    MAY ( l $ description $ manager ) )
  objectClasses: ( 1.3.6.1.1.1.2.7 NAME 'ipNetwork'
    DESC 'Abstraction of an IP network'
    SUP top STRUCTURAL
    MUST ( cn $ ipNetworkNumber )
    MAY ( ipNetmaskNumber $ l $ description $ manager ) )
  objectClasses: ( 1.3.6.1.1.1.2.8 NAME 'nisNetgroup'
    DESC 'Abstraction of a netgroup'
    SUP top STRUCTURAL
    MUST cn
    MAY ( nisNetgroupTriple $ memberNisNetgroup $ description ) )
  objectClasses: ( 1.3.6.1.1.1.2.9 NAME 'nisMap'
    DESC 'A generic abstraction of a NIS map'
    SUP top STRUCTURAL
    MUST nisMapName
    MAY description )
  objectClasses: ( 1.3.6.1.1.1.2.10 NAME 'nisObject'
    DESC 'An entry in a NIS map' SUP top STRUCTURAL
    MUST ( cn $ nisMapEntry $ nisMapName )
    MAY description )
  objectClasses: ( 1.3.6.1.1.1.2.11 NAME 'ieee802Device'
    DESC 'A device with a MAC address' SUP top AUXILIARY
    MAY macAddress )
  objectClasses: ( 1.3.6.1.1.1.2.12 NAME 'bootableDevice'
    DESC 'A device with boot parameters' SUP top AUXILIARY
    MAY ( bootFile $ bootParameter ) )
```

This lists all **objectclasses** known to the server. You can discover what all the OIDs mean using this great site or alternatively this terrific site.

Up Arrow

Chapter 4 Installing LDAP

Go to Arrow

[an error occurred while processing this directive]
[an error occurred while processing this directive]
[an error occurred while processing this directive] [an error occurred while processing this directive]
[an error occurred while processing this directive] [an error occurred while processing this directive]