LDAP Schema Design

Andrew Findlay

Skills 1st Ltd

February 2005

andrew.findlay@skills-1st.co.uk

http://www.skills-1st.co.uk/

Synopsis

It is possible to make one LDAP directory serve many applications in an organisation. This has the advantage of reducing the effort required to maintain the data, but it does mean that the design must be thought out very carefully before implementation starts.

LDAP directories are structured as a tree of entries, where each entry consists of a set of attribute-value pairs describing one object. The objects are often people, organisations, and departments, but can be anything at all. Schema is the term used to describe the shape of the directory and the rules that govern its content.

A hypothetical organisation is described, with requirements for “white pages” directory service as well as a wide range of authentication, authorisation, and application-specific directory needs. The issues arising from the LDAP standards are discussed, along with the problems of maintaining compatibility with a range of existing LDAP clients.

A plan is proposed for the layout of the directory tree, with particular emphasis on avoiding the need to re-organise it later. This involves careful separation of the data describing people, departments, groups, and application-specific objects. A simple approach to entry design is shown, based on the use of locally-defined auxiliary object classes. The effects of schema design on lookup performance are discussed. Some design tricks and pitfalls are presented, based on recent consulting experience.

1 LDAP

LDAP can be used to access information describing people, organisations, roles, services, and many other sorts of entity. It is a standard and widely-implemented protocol, which makes it extremely valuable for integrating multiple applications that need to share common data.

It is important to understand the LDAP data model when considering schema. It is different from the relational model used by most well-known database systems, and this affects the way LDAP systems are designed and used.

Strictly speaking, LDAP is a *protocol* - the Lightweight Directory Access Protocol. It is not a *database* or even a *directory* although the term *LDAP directory* is often used to describe a directory service that is accessed using the LDAP protocol. LDAP derives from the X.500/ISO-9594 standards, and was originally intended as a simplified protocol for small computers to use when accessing X.500 systems. In recent years, LDAP has expanded and is now just as complex as X.500 but it still shares the same data model and part of the same distributed-service model. [RFC3377]

1.1 The LDAP data structure

An LDAP Directory stores information in a tree structure known as the Directory Information Tree (DIT). The nodes in the tree are directory entries, and each entry contains information in attribute-value form. Some attributes may have multiple values; others are restricted to a single value. The set of attributes that may be present in an entry is determined by the *objectClass* attribute,which is always present. *objectClass* is a multi-valued attribute and each value defines a set of mandatory and/or optional attributes.

Each node in the DIT has a name called the *Relative Distinguished Name* (RDN) which is unique among the peer nodes under its parent. It also has a globally-unique name called the *Distinguished Name* (DN) made up from the name of the node itself plus the names of all its superior nodes up to the root of the DIT.

1.2 The LDAP Distributed Service model

X.500 was conceived as a global directory service. As such, it was expected to hold hundreds of millions of entries and be managed by thousands of different organisations. This led to a service model based on many co-operating servers known as DSAs (Directory System Agents). Each DSA can hold data from one or more arcs of the DIT, and is provided with “knowledge” so that it can direct queries for non-local data to the appropriate place. Queries are expected to outnumber updates by a very large factor, and absolute data consistency across the DIT is not supported, so data can be replicated easily for performance and resilience.

An LDAP server is effectively a DSA and it follows the same rules, though LDAP does not have such a good distributed-service model for very large networks. Most current LDAP deployments are limited to operation within a single organisation, so although data replication is commonly used, the other features of the distributed service model do not get so much exercise.

X.500 and LDAP have many similarities with DNS in terms of data model and service model, but the directory systems are capable of more complex operations.

2 Requirements

When considering an LDAP deployment that is to serve more than one application, it is important to get the widest possible view of the host organisation and its future needs. This is because it is easy to change the shape of the DIT at the design stage but very hard once data has been loaded and applications are running.

2.1 The Example Organisation

Consider a hypothetical organisation called **The Example.** It has a domain name **example.org** and wants to integrate several systems that handle data relating to people. The services required are:

- White Pages searches for People and Roles, from several existing user interfaces.
- Authentication - simple (e.g. web portal) and with extra data (e.g. Samba for Windows clients and Unix NSS/PAM)
- Authorisation - groups of various sorts, permissions, access-control for particular applications
- Storage of application-specific data - config, personalisation, mapping tables
- Integration of mail-system configuration with White Pages directory

It is certainly possible to meet all these requirements with a single LDAP-based directory, though it will need some careful thought about how the data is to be managed.

3 LDAP issues and design principles

3.1 Avoid renaming things

Once an object has been created in an LDAP directory it is likely that the name of that object will get stored in other objects, or even outside the directory entirely. This suggests that objects should never be renamed, which in turn restricts the choice of how they should be named in the first place.

Most of the examples in the X.500 and LDAP standards documents use “natural” names for objects. Thus the object representing a person is usually shown with an RDN based on the person's name. This leads to two problems: what to do if the person changes their name, and how to cope with several people who happen to share the same name. The “standard” approach to the name-clash problem is to use an extra attribute in the RDN to remove the ambiguity. Thus, multiple Fred Smiths end up with RDNs of the form “cn=Fred Smith + uid=67354”. This does not solve the name-change problem though, so I recommend that all entries representing people or other entities that might change name should be given RDNs containing only the *uniqueIdentifier* attribute: the value can be a simple serial number, which does not have to mean anything outside the directory.

There is a potential problem with this approach too: some directory user interfaces use the RDN value to title the entry display. A meaningless serial number would look odd when used for that purpose. Fortunately this behaviour is now rare, and the standards provide an attribute that is explicitly intended for the job: *displayName*. It is thus a good idea to include *displayName* in every directory entry.

3.2 Do not modify the standard schema definitions

The set of attributes and object classes described in the LDAP RFCs sometimes appear to be rather awkward. This is not surprising, as they were derived from many different postal and telecomms standards, and new ones were created whenever the early directory service developers found a new requirement. As a result, people are often tempted to “tidy up” or “correct” the standard schema. This is a Bad Idea, as knowledge of the standard attributes is embedded in many widely used products. To make matters worse, many product authors have created their own overlapping or conflicting attribute sets to enrich the “addressbook experience”.

Fortunately, the better LDAP clients can be configured to use whatever attributes you want so it is often best to ignore the more broken parts of the standard set and create your own replacements. Some clients cannot cope with this, so it is worth finding out what you have to support and how flexible it is.

3.3 Beware the Structural Object Class

Every entry has a *Structural* object class. This defines its fundamental characteristics, and cannot be changed once the entry has been created.

If you define new attributes, or want to combine existing ones in new ways, you will need to create new object classes that allow this to happen. Most of the examples in the standards show this being done by subclassing existing object classes, but this creates problems. The LDAP standard diverged from X.500 just before the development of X.500(1993), and has adopted parts of the later versions piecemeal. One of the later innovations was to split objectclasses into three types and to tighten the rules on how they could be combined. LDAP servers have now started to enforce these rules, and as a result the over-use of structural object classes can seriously limit the flexibility of a directory service.

I recommend that you pick a well-understood structural class for each type of entry, and then add *Auxiliary* object classes where needed. The rules for combining these are much simpler, and they can be added to existing entries or removed if desired. Some reasonable structural classes are:

Object	***Structural class***
Person	inetOrgPerson
Role	organizationalRole
Department	organizationalUnit
Organisation	organization

4 DIT Layout

4.1 Top levels

The standards place very few restrictions on the form of the DIT, though if it is to be used with common client software there are some conventions that should be observed. There are two common forms used for the higher levels in the tree: geographic and domain-based. The geographic form was used in all examples in the original X.500 standards, as it is most suitable for global-scale “white pages” searching. Domain-based trees were introduced later, partly so

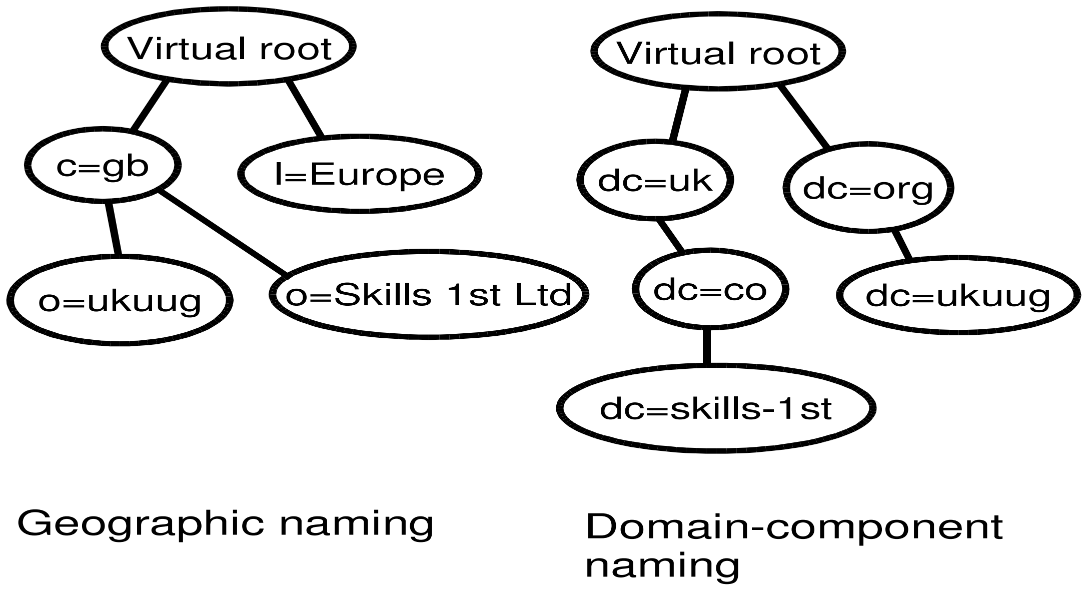

Figure 1: Higher levels of the DIT

that installers could avoid dealing with country-level registration authorities and partly to provide a simple way to map Internet domain names into the DIT. The two forms are shown in Figure 1.

For an isolated directory system it makes little difference which form is chosen, though it is always wise to choose a base name that will not clash with any other directories that might get linked in future. The two forms can co-exist under the same virtual root if required.

For most “internal” directory services, the Domain Component tree is the better choice. There is a direct mapping between DNS names and the upper levels of the tree, which allows the address of the LDAP server to be stored in the DNS and can make client configuration simpler. I generally recommend carrying the DC names down into the “local” part of the tree, as this makes it easier to divide the tree among several servers if required.

Our Example Organisation owns the domain name **example.org** so we will choose **ds.example.org** as the base for the directory service. This translates to an LDAP suffix of **dc=ds,dc=example,dc=org**

4.2 Lower levels

4.2.1 Where to put the people

When building a white-pages directory service, there is a tremendous temptation to build a DIT that mimics the structure of the organisation itself. This is what the standards seem to have been designed for, and what most of the examples show. It also seems like a good way to achieve delegated management: if all the people in the Customer Service Department are listed under that department's own entry, it should be easy to hand over control to the departmental administrator. Unfortunately, the one constant feature in most organisations is change. Departments come and go, they get re-named and merged, yet most employees survive the changes and continue to do very similar jobs. If we have to re-name the departmental entries every year or so, the people-under-departments model means that we are also re-naming each person's entry and we already know that this is Bad.

The usual solution to this problem is to list all people under a single node in the DIT. Thus, the Example Organisation might create a node **dc=people,dc=ds,dc=example,dc=org** for this purpose. Most organisations have fuzzy edges, as the exact distinctions between “staff”, “contractors”, “suppliers”, “business partners”, “customers”, “visitors”, etc are not rigid or constant. It therefore makes sense to group all types of person into one place and to make any necessary distinctions by adding attributes to individual entries, or by using group objects.

Using the convention established in section 3.1 above, the entries themselves will have opaque names so a typical person entry might have the DN:

```
uniqueIdentifier=67325,dc=people,dc=ds,dc=example,dc=org
```

4.2.2 Representing the organisation

The organigram-style tree still has its uses, even if we are not going to put entries in it for actual people. In a white pages context, someone searching the directory does not always know the name of the person they need to contact: they just want the number for “The IT Helpdesk” or “The Finance Office”. After a major re-organisation, many people will not even know the name of the department they need to contact, so browsing through a list might be the only option.

To serve these requirements, it is possible to add an arc (subtree) of the DIT called **dc=departments** or similar. The structure of the organisation can be represented using **organizationalUnit** and **organizationalRole** nodes. Where possible, the contact information listed in these entries should be role-specific rather than person-specific. Thus it would be better to show an address of the form **finance@example.org** than to give the addresses of the people actually working there today. If it is necessary to make a link to actual people, the *roleOccupant* attribute can be used, which is effectively a pointer to an entry in the *dc=people* part of the tree.

4.2.3 Groups

People form themselves into groups for all sorts of purposes. The directory might have to represent committees, mailing lists, clubs, people with particular powers or privileges, etc. Some groups might have defined roles in the organisation (Departmental IT Support Officers) but others might only have a meaning to one person (Jane's lunch group).

Groups are normally represented by LDAP objects with the *groupOfNames* class. In these objects, group members are represented by *member* attributes, whose values are the DNs of the members themselves. The *member* attribute can obviously have multiple values. This is a simple scheme, but it does have two significant problems:

- The *groupOfNames* class makes the *member* attribute mandatory. This means that it is not possible to create an empty group, which makes data management harder.
- It is not possible to attach any extra information to the DN value to show *why* that person is a member. This is awkward for committees, where you might need to show who is Secretary, and who is an observer.

There are ways around these problems, but they involve changing the object class and structure of the objects. This might break compatibility with existing applications, so a careful analysis of the requirement is essential.

Groups can be placed in any part of the DIT, but the requirement for a shared set of groups is so common that it is usually worth creating an arc specifically for this purpose. Depending on the complexity of the organisation, this might be a single **dc=groups** node, or it might be a subtree divided into different categories.

4.2.4 Application-specific data

Most software applications need to store configuration data somewhere. For networked applications there are advantages to storing this in the LDAP directory:

- Multiple servers can easily access the configuration
- Configuration can be managed with generic LDAP browser/editor applications
- By storing configuration in LDAP, it may be possible to make the application servers effectively “dataless” so they can be replicated easily and do not need to be backed up.
- Many network applications use LDAP for authentication. Placing the authorisation data in the same sort of store can help with consistency and efficiency.

There should be one arc of the DIT for each application. The management of these subtrees can be delegated to the application managers. The LDAP service itself is a network application, so it should have a subtree in which to store groups used for access control etc.

4.3 DIT Overview

The DIT for the Example Organisation now has arcs (subtrees) for several specific purposes, as shown in Figure 2.

5 Attributes and object classes

All information within a directory entry is stored as attribute-value pairs. The set of attributes that can appear in a given entry is set by its *objectClass*. Some of the attributes are mandatory, but most are optional. New attributes can be defined if necessary, and new object classes can also be added to permit the new attributes to appear in entries.

The standards already define a very wide range of attributes, so an important part of the schema design process is to decide which ones will actually be used. This section proposes a reasonable subset to start with.

The common attributes divide into several broad groups:

- Naming attributes: these hold the actual names of the real-world object being represented, and are commonly used for searching.
- Descriptive attributes: photos, textual descriptions.
- Postal attributes: these deal with physical location and the delivery of physical objects.
- Telecommunication attributes: phone numbers, e-mail addresses etc.
- Authentication attributes: usernames, passwords, unique identifiers, data for Samba and Kerberos, X.509 certificates.
- Management attributes: owners, managers, access-control lists.

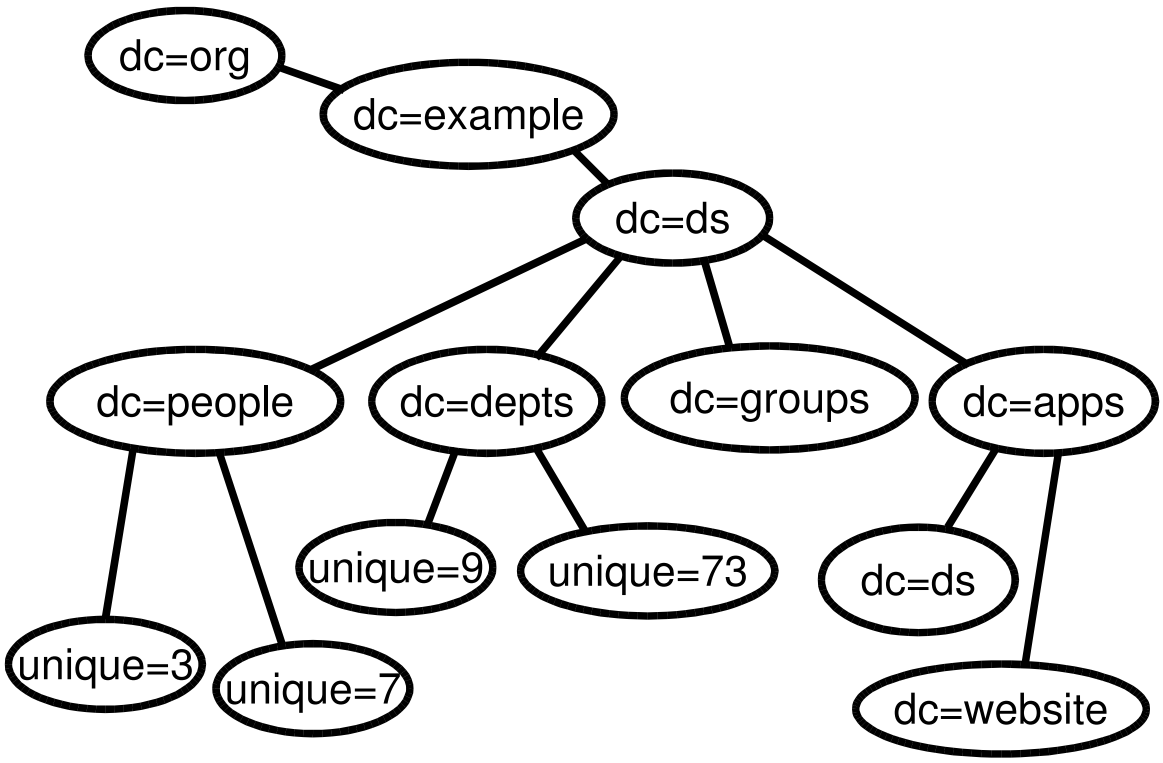

Figure 2: DIT Overview

5.1 Naming attributes

Most of these are derived from the generic *name* attribute type. This gives them a length limit of 32768 characters, the UTF-8 character set, and a set of case-insensitive matching operations.

In the descriptions below, paragraphs in *italics* are taken directly from the LDAP or X.500 standards documents.

5.1.1 cn (Common Name)

> *This is the X.500 commonName attribute, which contains a name of an object. If the object corresponds to a person, it is typically the person's full name.*

cn is often multi-valued, to aid searches in the directory, e.g.:

```
cn: Brian Stanley Walker
cn: Mr B S Walker
cn: Paddy Walker
```

cn is mandatory in person entries, and is one of the most important attributes as it is commonly specified in white pages searches. *cn* is also used in entries that do not refer to people: rooms, roles, printers, groups etc.

Note that some information appearing in *cn* will also appear in other attributes, particularly *sn*, *givenName, personalTitle,* and *displayName.* This use of *cn* is very important to support searching, particularly where names can include characters that are not available on all keyboards, e.g. this Polish name:

```
cn: Minka Krzyžańska
cn: Minka Krzyzanska
```

In this case, the CN attribute holds the "real" name and also a simplified form that can be typed without attention to diacritical marks. The simplified CN makes searching easier, and the real name is available for display in the *displayName, sn,* and *givenName* attributes.

5.1.2 displayName

When displaying an entry, especially within a one-line summary list, it is useful to be able to identify a name to be used. Since other attribute types such as 'cn' are multivalued, an additional attribute type is needed. Display name is defined for this purpose.

This is a single-valued attribute. It is strongly recommended that this attribute be filled in with the preferred form of name (which should also appear as a value of *cn*). All appropriate diacritical marks should be included. *displayName* should also be used on all non-person entries so that there is a consistent way to find the "proper" name of an object.

5.1.3 sn (Surname)

This is the X.500 surname attribute, which contains the family name of a person.

sn is mandatory in person entries and can hold multiple values. The standards betray a Western cultural bias here, and I suggest that where a person does not have distinct "given" and "family" names, their entire name should be placed in both *cn* and *sn* attributes.

Note that multi-part surnames such as *van der Vorst* should be handled as a single value of *sn*. The use of multiple values may be appropriate where someone has recently changed their name, but this is risky as many applications assume a single value for the *sn* attribute and may behave unpredictably in the presence of multiple values. Name-changes are better handled by adding values to the *cn* attribute.

The value stored in the *sn* attribute should be the "correct" form with all diacritical marks as used by the person themselves.

5.1.4 givenName

The givenName attribute is used to hold the part of a person's name which is not their surname nor middle name.

A less-commonly used attribute. Provides better normalisation of data than *cn* and costs little to populate so worth using in most cases. As with *sn*, should be the "correct" form including diacritical marks where appropriate.

5.1.5 personalTitle

The Personal Title attribute type specifies a personal title for a person. Examples of personal titles are "Ms", "Dr", "Prof" and "Rev".

This comes from RFC1274 and has a length limit of 256 characters. It allows multiple values, but where an individual has multiple titles it is best to place them all in one value as this preserves the order, e.g. “Eur Ing Dr”.

5.1.6 o (Organisation Name)

> *... a string chosen by the organisation (e.g. “Scottish Telecommunications plc”) Any variants should be associated with the named organisation as separate and alternative attribute values.*

The use of multiple values is appropriate in entries describing organisations, but may not be as relevant when using an organisation attribute in a person entry.

Length limit 64 characters in X.520. RFC2256 redefines *o* under the *name* superclass, so in LDAP it gets a limit of 32768 characters.

5.1.7 ou (Organisational Unit Name)

> *The Organisational Unit Name attribute type specifies an organisational unit. When used as a component of a directory name it identifies an organisational unit with which the named object is affiliated.*
>
> *The designated organisational unit is understood to be part of an organisation designated by an Organisation Name attribute. It follows that if an Organisational Unit Name attribute is used in a directory name, it must be associated with an Organisation Name attribute.*
>
> *An attribute value for Organisational Unit Name is a string chosen by the organisation of which it is part (e.g. ou=”Technology Division”). Note that the commonly used abbreviation “TD” would be a separate and alternative attribute value.*

Length limit 64 characters in X.520. RFC2256 redefines *ou* under the *name* superclass, so in LDAP it gets a limit of 32768 characters.

5.2 Descriptive attributes

5.2.1 jpegPhoto

> *Used to store one or more images of a person using the JPEG File Interchange Format [JFIF].*

Size limit 250000 bytes. If this attribute is used, it tends to dominate the size of an entry and can have an impact on performance so it should be kept as small as possible.

5.2.2 description

> *This attribute contains a human-readable description of the object.*

Size limit 1024 characters. More likely to be useful in application-specific entries or group entries than in those related to people.

5.3 Postal attributes

X.500 inherited a lot of these from earlier ISO and CCITT standards. Unfortunately they do not form a complete and unambiguous set. I suggest ignoring most of them (*locality, streetAddress, stateOrProvinceName,* etc) in favour of country, postcode, and the full postal address.

5.3.1 postalAddress

> *(From X.520(1988)) ... address information required for the physical delivery of postal messages ...*
>
> *... limited to 6 lines of 30 characters each, including a Postal Country Name. Normally the information contained in such an address could include an addressee's name, street address, city, state or province, postal code, and possibly a Post Office Box number depending on the requirements of the named object.*

The implication here is that this attribute should fit onto a small address label or be usable directly on a letter to be posted in a window-envelope.

5.3.2 postalCode

> *The postalCode attribute specifies the postal code of the named object. If this attribute value is present it will be part of the object's postal address.*

Length limit 40 characters. (Longer than a single line in a postal address!) Note that the *postalAddress* attribute also contains the postal code, but it is useful to hold a copy separately as postcodes are useful in many ways besides addressing letters. (This is less true outside the UK, as in most countries the postcode only identifies the nearest town, whereas UK postcodes are much more specific)

5.3.3 c (Country Name)

> *This attribute contains a two-letter ISO 3166 country code*

This attribute may only carry a single value. Note that there is also a *friendlyCountryName* attribute which can be used to hold one or more human-readable country names. In most cases it is better to use the two-letter ISO3166 code in the *c* attribute and leave "user friendly" presentation to the user-interface.

5.4 Telecommunications attributes

This is a vast set, many appearing in both "work" and "home" forms. The subset shown here is enough for most white-pages applications.

5.4.1 mail (RFC 822 mail address)

> *RFC 1274 uses the longer name 'rfc822Mailbox' and syntax OID of 0.9.2342.19200300.100.3.5. All recent LDAP documents and most deployed LDAP implementations refer to this attribute as 'mail' and define the IA5 String syntax using using the OID 1.3.6.1.4.1.1466.115.121.1.26*

This is the normal electronic-mail address, which must be given in full RFC822 format (e.g. andrew.findlay@skills-1st.co.uk). Note that the attribute can have multiple values and no order of preference is implied: I suggest a limit of one value, as there is no way to express an order of preference among multiple values. Note also that in keeping with RFC822 the attribute only allows the 7-bit IA5 character set.

Size limit 256 characters.

5.4.2 telephoneNumber

> *Telephone numbers are recommended in X.520 to be in international form, as described in E.123 [15].*
>
> *Example: +1 512 305 0280*

Telephone numbers follow the *printableString* syntax so a fairly wide character set is accepted. This allows things like:

```
+44 207 123456 x9876
```

A local convention is needed if extension numbers are to be included.

Note that there are also *homeTelephoneNumber, facsimileTelephoneNumber,* and *mobile* attributes with similar characteristics.

Size limit 32 characters.

5.5 Authentication attributes

LDAP standards define a large set of attributes for describing computer accounts and for the storage of X.509 certificates. Only the simpler ones are mentioned here.

5.5.1 uid (User Identifier)

> *The Userid attribute type specifies a computer system login name.*

Length limit 256 characters. Note that this is not a UID in the Unix sense: it is a username. If Unix account data is to be held in an LDAP format then *uidNumber* should be used for the numeric UID. Note also that this is a case-insensitive attribute so the usernames *root* and *ROOT* refer to the same object which is different from the traditional Unix behaviour.

5.5.2 userPassword

> *Passwords are stored using an Octet String syntax and are not encrypted. Transfer of cleartext passwords are strongly discouraged where the underlying transport service cannot guarantee confidentiality and may result in disclosure of the password to unauthorized parties.*

The definition quoted here comes from RFC2256. However, RFC2307 expands the definition to permit a range of hashed passwords to be stored. Note that hashed passwords cannot be used with all authentication mechanisms so the choice is not straightforward.

This attribute can hold multiple values, each of which is limited to 128 characters.

Access-control is normally set on this attribute to prevent anyone from reading it (including directory managers and the owner of the entry).

5.5.3 Unix account attributes

There is a set of attributes known as the “NIS Schema”, which directly map fields from the Unix passwd file and group file. These should be included where the LDAP directory is acting as a Network Information Service (NIS). See Reference 1 for more information on this.

Attributes include *uidNumber, gidNumber, gecos, homeDirectory, loginShell,* and a whole set to describe password expiry rules. See reference 2 [RFC2307].

5.5.4 Samba and Windows attributes

Later versions of Samba can use the LDAP directory to store Windows Domain information. This allows Windows accounts to be closely linked to white pages entries and also to Unix accounts. Again, Reference 1 has more detail.

5.5.5 X.509 attributes

Directories are an important part of any Public Key Infrastructure (PKI). Attributes are provided for the storage of certificates, revocation lists etc.

5.6 Management attributes

5.6.1 title

> *The title attribute type specifies the designated position or function of the object within an organisation.*
>
> *Example: title=”Manager, Distributed Applications”*

Length limit is 64 characters in X.520. RFC2256 redefines *title* under the *name* superclass, so in LDAP it gets a limit of 32768 characters.

5.6.2 owner

> *The* owner *attribute type specifies the name of some object which has some responsibility for the associated object. The value is a Distinguished Name*

From X.520. This attribute is useful in conjunction with Access Control Lists to delegate control of directory entries to particular people.

5.6.3 member

The *member* attribute is used in entries defining groups. It has Distinguished Name syntax, so each value is effectively a pointer to another entry in the directory. This is a multi-valued attribute. Note that the standard *groupOfNames* object class makes the *member* attribute mandatory. As attributes cannot have empty values, this effectively requires all groups to have at least one member at all times. To avoid the problems this would create, I suggest creating a new objectclass that makes *member* optional rather than mandatory.

Group entries are often important when defining access control rules. They are also useful when expressing authorisation policies for applications.

5.6.4 Access Control Lists

In most directory systems there is a need to control who can update entries, and often there is also a need to control who can read certain data (*userPassword* attributes as a minimum). This is normally done using Access Control Lists (ACLs), but unfortunately the structure of these has not been standardised and it varies from one LDAP server product to another.

ACLs are often stored as attributes in the DIT, affecting entire subtrees (IBM Tivoli Directory Server and Oracle Internet Directory). In some cases they are stored outside the DIT along with rules that specify what entries they affect (OpenLDAP).

5.7 Object Classes

Our Example Organisation will need to define some new object classes. This requires a unique Object Identifier (OID) to act as a base. OIDs can be obtained from IANA at http://www.iana.org/cgi-bin/enterprise.pl or from many national assignment bodies.

In the UK, the rules for assigning OIDs at a national level are given in BS 7453 Part 1, section 6. For organizations registered as a company under the provisions of the Companies Act 1985, section 6.1.1 states that the company "is deemed to be assigned" an object identifier that starts with 1.2.826.0.1 for England and Wales, 1.2.826.0.2 for Scotland, and 1.2.826.0.3 for Northern Ireland, and has as its next component "the integer value of the numeric component of the registered number". So Skills 1st Limited, having the registry number 3458854 in England, has the implicit OID 1.2.826.0.1.3458854

Assuming the Example Organisation has been assigned the OID 1.3.6.1.4.1.98546 it might choose to make assignments like this:

OID	***Purpose***
1.3.6.1.4.1.98546.1	Directory Service
1.3.6.1.4.1.98546.1.1	Attribute Types
1.3.6.1.4.1.98546.1.2	Object Classes
1.3.6.1.4.1.98546.1.2.1	exampleObject
1.3.6.1.4.1.98546.1.2.2	examplePerson
1.3.6.1.4.1.98546.1.2.3	exampleGroup

5.7.1 exampleObject

The *exampleObject* class will be applied to almost every entry in the directory. Its purpose is to require *displayName* to be added, and to permit *uniqueIdentifier* because we use that so much in RDNs:

```
objectclass ( 1.3.6.1.4.1.98546.1.2.1 NAME 'exampleObject'
    DESC 'example.org objects'
    SUP top AUXILIARY
    MUST displayName
    MAY uniqueIdentifier )
```

If requirements change later, it would be possible to add more permitted attributes to this object class and they would automatically become available in all the entries that use it. Adding extra mandatory attributes would be unwise, as existing entries would not have them, and thus the data already in the directory would not match the schema.

5.7.2 examplePerson

This class exists to allow attributes into “people” entries that are not already permitted by *inetOrgPerson*. Initially it is very simple, and just adds *personalTitle* and *ou* attributes:

```
objectclass ( 1.3.6.1.4.1.98546.1.2.2 NAME 'examplePerson'
    DESC 'example.org people'
    SUP exampleObject AUXILIARY
    MAY ( personalTitle $ ou ))
```

Note that *examplePerson* names *exampleObject* as its super-class. This means that all *examplePerson* entries must also conform to the requirements of *exampleObject* - they have to include a *displayName* attribute.

5.7.3 exampleGroup

This is a replacement for the standard *groupOfNames* class. Note that it is a structural class rather than an auxiliary one, as it will normally be used alone.

```
objectclass ( 1.3.6.1.4.1.98546.1.2.3 NAME 'exampleGroup'
    DESC 'example.org group'
    SUP top STRUCTURAL
    MUST displayName
    MAY ( member $ description $ owner ))
```

By making the *member* attribute optional we allow for empty groups.

6 Entries

We now have a set of attributes to choose from and some object classes to go with them. In this section we see how these are combined to make up typical entries.

6.1 People

Entries describing people will be based on the *InetOrgPerson* object class. We will also add the *examplePerson* class, and this brings *exampleObject* with it.

People entries will have RDNs of the form **uniqueIdentifier=37284** so this must also appear in the entry itself.

The initial set of attributes for “person” entries is:

Attribute	***Notes***	***Mandatory***
objectClass	inetOrgPerson examplePerson	Y
uniqueIdentifier	Single value, used as RDN	Y
cn	Likely to be multi-valued, e.g.: cn: Mr Brian Stanley Walker cn: Paddy Walker	Y
sn	Surname	Y
gn	Given Name(s)	N
personalTitle	Dr, Mrs, Prof, etc	N
displayName	The normal form of name used in written matter, e.g. “Sir Joseph Porter KCB” The value stored here should always be used rather than constructing a full name from other attributes.	Y
title	Job or role title, e.g. “First Sea Lord”	N
jpegPhoto	May be useful for ID verification. Keep it as small as possible.	N
mail	E-mail address.	N
telephoneNumber	Assumed to be the “office” number. All phone numbers to be in international form, e.g.: “+44 1763 273475”	N
mobile	Mobile phone number	N
postalAddress	Pre-formatted address for direct use on labels etc	N
postalCode	Post code	N
ou	Organisational Unit name (department/division etc) This is useful where a search by name returns several entries. Checking the department will often resolve the problem.	N

To this we add authentication and NIS attributes if the person will have a Unix or Windows account.

6.2 Roles

Roles appear in the Departments arc of the DIT, and are based on the *organizationalRole* object class. RDNs follow the convention established for “people” entries: **uniqueIdentifier=783263** (note that the standards only require these IDs to be unique among the immediate sibling entries, but it is probably best to make them unique across the entire directory).

Attribute	*Notes*	*Mandatory*
objectClass	organizationalRole exampleObject	Y
uniqueIdentifier	Single value, used as RDN	Y
cn	May be multi-valued, e.g.: cn: Finance Officer cn: Head Bean Counter	Y
displayName	The normal form of name used in written matter, e.g. “Finance Officer”	Y
mail	E-mail address. Where possible this should be a proper “functional” address like finance@example.org rather than the address of the person who happens to fill the role.	N
telephoneNumber	All phone numbers to be in international form, e.g.: “+44 1628 782565”	N
postalAddress	Pre-formatted address for direct use on labels etc	N
postalCode	Post code	N
roleOccupant	DN of *person* entry where the person filling the role is known in the LDAP database.	N
ou	Organisational Unit name (department/division etc) Although this information may also appear in the full DN of the entry, it make searching much easier if it is repeated in the entries.	N

6.3 Organisational units

Departments, divisions, and similar entities are represented in the organisational structure part of the DIT.

Attribute	*Notes*	*Mandatory*
ou	Organisational unit name (may be multi-valued)	Y
o	Organisation name (may be multi-valued). This is useful if multiple organisations are to be represented in the tree, as it makes searching and display easier.	N
displayName	Required because *ou* may be multi-valued	Y
mail	E-mail address for the “front desk” function	N
telephoneNumber	Main switchboard or enquiry number	N

Attribute	*Notes*	*Mandatory*
postalCode		N
uniqueIdentifier	Used as RDN of entry so effectively mandatory. Only needs to be unique among the entry's immediate peers.	N

6.4 Data shared between applications

Organisations often have data sets that are used by several applications. One example I came across recently was a list of countries along with their ISO 3166 codes, pictures of their flags, and other data. This sort of read-mostly dataset is a very good candidate for inclusion in a corporate directory.

The country data can be included by defining an arc of the DIT such as **dc=countries,dc=apps,dc=ds,dc=example,dc=org** and a suitable objectclass for the entries. Each entry would contain attributes such as:

Attribute	*Notes*	*Mandatory*
c	ISO 3166 two-letter country code	Y
displayName	Full name of the country	Y
exampleThreeLetterCC	ISO 3166 three-letter country code	Y
friendlyCountryName	The full name of the country, possibly including the name in multiple languages	Y
jpegPhoto	JPEG image of national flag. Note that if PNG images are to be stored, a new attribute must be defined for the purpose.	Y

LDAP servers are capable of doing searches against almost any attribute, so a variety of lookups are possible on such a dataset.

6.5 Authentication and authorisation

Networked applications frequently use LDAP to support authentication. In the simplest form, they present the username and password supplied by the user in an LDAP Bind operation: if the Bind succeeds, this proves that the password is correct. Stronger schemes involving Kerberos or client-side X.509 certificates can also be supported.

Authorisation can be more complex: this is the job of working out what the user is permitted to do once they have proved their identity. Most applications define a set of roles, each with permission to do certain things, and then assign users to roles. This can be represented using multi-valued attributes in the LDAP directory.

Suppose our Example Organisation has a web portal with a content management system. This supports several levels of access:

- Read-only access to public data
- Read-only access to data in certain defined categories

- Author access to create new content
- Editor access to modify content created by others
- Manager: can set access permissions for others

The first level is the default for all users. All the rest need to be represented in the directory.

There are two basic schemes that we could choose here:

1. Put a permissions attribute in the user entry, and give it one value for each permission that the user has been given.
2. Create an entry somewhere for each permission, and use a *member* attribute to list the users who have that permission.

The first scheme has the advantage that a single directory read provides all the authentication and authorisation data for a given user. It also deals with user deletion very easily. However, it does mean that the web-portal manager must be given access to change user entries, and it creates the risk that a user or user-entry manager might be able to change their own level of access on the web portal.

The second scheme collects all the permissions data for the application into a dedicated part of the DIT. This makes it easier to manage and secure, but does require some extra clean-up effort when users are deleted. It also allows for permissions data to be stored in a completely separate LDAP server if necessary. I normally recommend the second scheme.

Authorisation data should go in the application-specific part of the DIT. In the web-portal example, we would probably create an entry:

```
    dc=authorisation,dc=portal,dc=apps,dc=ds,dc=example,dc=org
```

There would be one entry under that node for each permission that we need to represent.

An application wishing to check whether a user has some particular permission now performs these operations:

1. Find the user's DN: this will normally be done during the authentication phase
2. Derive the DN of the relevant permission entry from the permission name. This is usually a simple string concatenation.
3. Perform a “base object” search on that DN using the assertion that the user's DN is in the *member* attribute. If the search returns a result, then the user has the permission being tested.

7 Data management

The initial loading and ongoing management of directory data is too big a subject to cover here. However, it is important to note that the directory schema may have to be extended to hold “foreign keys” so that entries can be correlated against other databases.

Another important consideration is that of who will be allowed to update the directory data. It may be useful to allow people to update their own entries, but if this is done it is important to make sure that they cannot modify any attributes that would give them extra powers (such as changing their *numericUid* attribute to zero!) Where entry management is delegated to departmental administrators, some attribute will be required in each entry to show which administrator is responsible for it.

I have found the *owner* attribute to be useful here, and in some cases have also created an attribute such as *manager* which holds a DN pointing to a group entry. Then, anyone listed in that group will have management control over the entry that references it. Both *owner* and *manager* can be multi-valued, which adds to the flexibility. Of course, these schemes depend on the ability of your chosen LDAP server to support the required form of ACL, which they cannot all do.

Another useful trick is to avoid deleting entries when people leave: just set an attribute to say they are inactive. This can be useful where an audit trail is required but does introduce more complexity in ACLs.

8 Performance issues

The layout of the DIT and the contents of entries can have an effect on the performance of the LDAP server. It is important to consider how the data will be used, and to design efficient searches. For example, if a client program needs to find a Unix-using person whose surname is Jones it is best to base the search at the root of the **dc=people** tree and to include suitable object classes in the search string:

```
    (&(sn=Jones)(objectClass=person)(numericUid=*))
```

Similarly, a search for application specific data can avoid the (probably very large) “people” arc, and limit its search to the subtree dedicated to the application itself.

LDAP servers can be optimised for different forms of query by adjusting indexing and caching parameters, and in a large installation it is critical to get this right. Unfortunately, some LDAP clients insist on issuing searches for every attribute in the book at once. Early versions of Evolution were particularly bad in this respect, and they still cast their net rather widely:

```
    (&(mail=*)(|(mail=andrew*)(|(cn=andrew*)(sn=andrew*))
        (displayName=andrew*)(sn=andrew*)))
```

Where performance of particular applications is critical, it may be necessary to replicate some or all of the DIT to dedicated servers with optimised indexing. The DIT layout proposed here allows for several variations of this idea.

9 Summary

When building an LDAP directory it is important to consider all requirements and to plan carefully at the start, as re-organising the Directory Information Tree later is very difficult. Organisations change constantly, and this must be allowed to happen without requiring major re-work in the directory.

This paper presents a suggested DIT layout and a set of objects to represent people, roles, and other common directory entries. These are designed to be extensible to cope with future requirements.

There is great benefit from sharing data across applications by placing it in the directory, but careful design is needed and bespoke management tools will probably be required.

10 References

1. *Security With LDAP*, Findlay, Andrew, UKUUG Winter Technical Conference, London, February 2002. http://www.skills-1st.co.uk/papers/security-with-ldap-jan-2002/
2. RFC2307 An Approach for Using LDAP as a Network Information Service, March 1998
3. RFC3377 Lightweight Directory Access Protocol (v3): Technical Specification, September 2002

~andrew/docs/papers/ldap/schema-design-feb-2005