The LDAP Directory Schema: a Guide for the Newcomer

Dr. Giovanni Baruzzi
giovanni.baruzzi@syntlogo.de

1 Abstract

We store information in an LDAP directory using Attributes and we group attributes in objects. These objects are built on a blueprint, the “objectClass” where we define what is in and what is not. Finally we store the definition of the attributes and object classes into a special object called “schema”.
As we start with LDAP, the schema contains already attribute and class definitions which have been defined after a careful standardisation process, but often we need to extend these definitions to store information specific to our project and not included in the standard schema.

The goal of this tutorial is to give instructions and discuss the best practices of how to extend an LDAP schema, getting a clear design that may support an organisation for many years.

Like the shape of a Directory Information Tree, even the Schema plays a major role to the success of an LDAP Project: with a proper Schema it is easy to identify the correct information for your purposes, knowing exactly the meaning of every Attribute and having a complete picture of your information model.
A Schema is the result of a design and a project can only be good as the design which underlines it.

The Newcomer could be overwhelmed by the task of designing a Schema. We want to follow him step-by-step showing him the resources at its disposal, how to build a data model, list “the dos and don’ts” and finally how to code it in an LDIF file.

1.1 The Audience

This paper is dedicated to people who have to design and implement an LDAP Directory for the first time or have to redesign one that already exists.
We are going here to cover the case of a middle sized Organization. Larger organizations may face problems that go beyond our scope.

2 Introduction

2.1 Why do we need a good Schema design?

We store information in an LDAP directory using Attributes and we group attributes in objects. These object are built on a blueprint, the “objectClass” where we define what is in and what is not. Finally we store the definition of the attributes and object classes into a special object called “schema”.
As we start with LDAP, the schema contains already attribute and class definitions which have been defined after a careful standardisation process, but often we need to extend these definitions to store information specific to our project and not included in the standard schema.

The goal of this tutorial is to give instructions and discuss the best practices on how to extend an LDAP schema getting a clear design that may support an organisation for many years.

A good design is always better than a bad one, but in our case we have the additional challenge to share the design among more people and more organizations than most Data Base architects do.
In fact, a major difference between a relational DB and an LDAP Directory is the number of organizational units involved in a project: while a relational DB serves normally the interests of only one “customer”, a directory work often for multiple different organizations and has to deliver a service to all them.
It is unthinkable to publish a schema and be forced to change it just after a few months, just as the developer already have already begun coding for it.
This fact implies a few important things:

1. The proposed information model should meet the needs of our customers and not ours.
2. Once deployed, this model is going to be difficult to change, because every change should leave the interface towards the customer unchanged or have to be agreed with all the affected users again.
3. To be successful, a Design has to be clear and consistent, just because it has to be embraced by every our customer.
4. The aim of a design should be clearly understandable by other people, because a directory is a long lived project.

Last but not least, a well designed Directory will be a success factor for our organization.

3 From the White Pages to a Business Model

Although this was their primary scope when they had been first designed, Directories are only sometimes used to locate people or to get in touch with them: most of the time they are used to identify users, authenticate them and to check if they have the entitlements to access IT Resources.

But we are still dealing with a model containing Persons' and Application objects even though the accent has shifted from white pages to access control. The directory contains those object inside a typical hierarchical structure and connected through implicit relationships and traverse links. The result is a complex graph modelling the organization, its users, the applications and the way they relate the one to the other.

The design has two facets: the first is the design of the tree itself, the Directory Information Tree (see my Tutorial for the 1st LDAP Conference on LDAP, Köln 2007) and the second, the Schema, the description of the information contained in the leaves and containers, is the object of this Tutorial.

The Schema ignores the relationships of the object in the tree and concentrates on the object himself, listing the properties that characterize it and the way they are encoded.

We are going to discuss the single objects like "user", "application", "organizational unit" or "group", but inserting the single descriptions inside a more complete picture and introducing a Reference Business Case, to give more depth to the single examples.

3.1 The Problem

A customer wants the redesign the concept of his customers' portal. He wants the new portal to incorporate state-of-the-art technologies, able to support Single Sign On and open to future developments.
He wants also the customers themselves, who are Organizations, and not private persons, be able to perform themselves more administrative tasks, envisaging a new role for the users: the role of a delegated administrator.

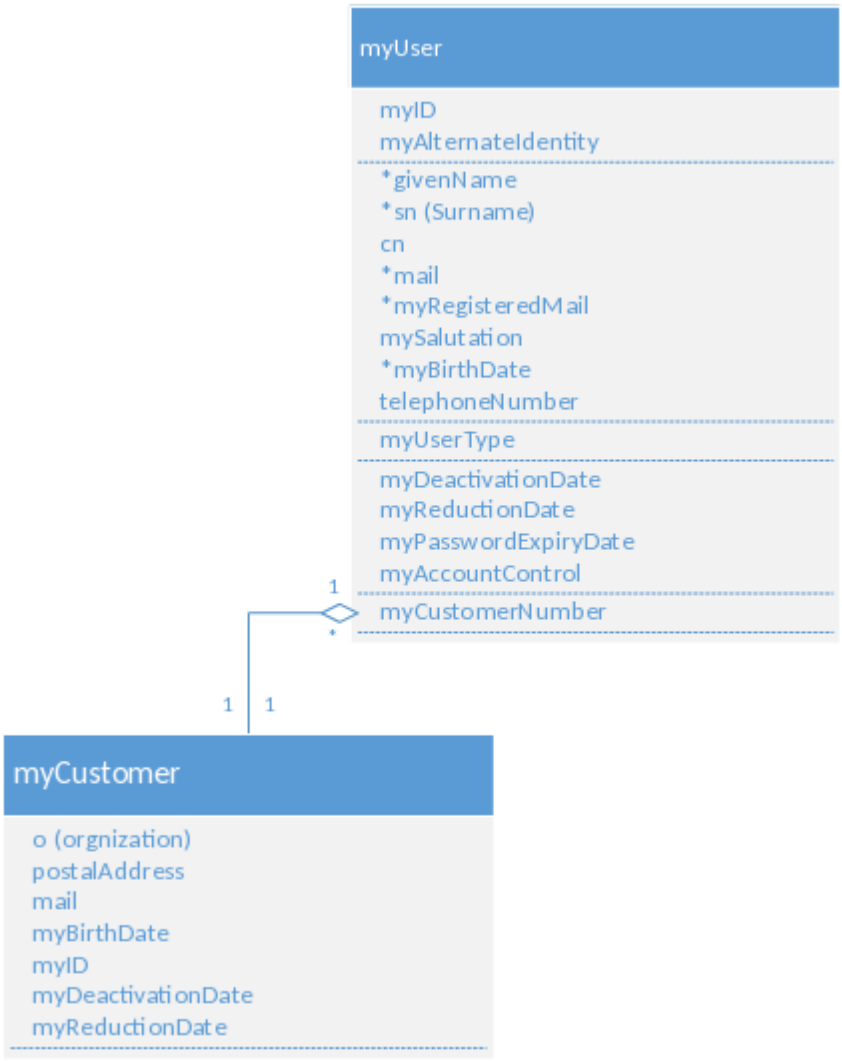
For the rest the requirements are to be more than skinny. "We need a new customer portal" and "we need an Identity Management to support them all, covering our existing applications and open to future needs".

In this challenge we are able to see the strength of a good architect: he is able to ignore some of the requirements of the legacy applications and their constraints and concentrate on a clean model of the business case. Once you have understood the business model, it is going to be far easier

to take the design into the future, as the changes in the business world are normally slower that the times in the IT.

3.2 The Data Model

The data model for the reference problem is fairly simple: a Use Object with a customer number added. A Flag to identify the delegated administrators.
A few contact data for the customer.
The user has some basic contact data, a few attributes for the password and account control and a simple status information, telling us when an account is active or can be removed from the directory.


myUser
myID
myAlternateIdentity
*givenName
*sn (Surname)
cn
*mail
*myRegisteredMail
mySalutation
*myBirthDate
telephoneNumber
myUserType
myDeactivationDate
myReductionDate
myPasswordExpiryDate
myAccountControl
myCustomerNumber
1
*
1
1
myCustomer
o (orgnization)
postalAddress
mail
myBirthDate
myID
myDeactivationDate
myReductionDate

4 The Design Process for a Schema

In the rfc4512 we can read:

```
As defined in [X.501]:
    Object classes are used in the Directory for a number of purposes:
      - describing and categorizing objects and the entries that
        correspond to these objects;
      - where appropriate, controlling the operation of the Directory;
      - regulating, in conjunction with DIT structure rule
        specifications, the position of entries in the DIT;
      - regulating, in conjunction with DIT content rule
        specifications, the attributes that are contained in entries;
      - identifying classes of entry that are to be associated with a
        particular policy by the appropriate administrative authority.
```

As you will notice, the process of design a Schema is a deep attempt to use as much as possible the standard attributes and classes and only in extreme cases to introduce customised aspects.

- Do the standard attributes cover your needs?
- When you do need a custom attribute?
- Do the standard classes cover your needs? Not even if extended by some custom attributes?
- Which standard attributes can I still use in my custom class?

5 The available tools

To implement our project, we have some tools and capabilities at our disposal:
the standard attributes and classes available and the capability to extend them. Another capability of the design process is the determination of the Directory Information Tree, but this is outside the scope of this tutorial and can be read from my previous contribution at the LDAPCon 2007.

5.1 The standard attributes

It is the most basic good practice to begin to use them before defining new, custom attributes and object classes, but to well use them you have to know them well. We review the most used of them, starting with the syntaxes and proceeding to attributes and object classes.

The first Attribute definitions were made in 1993, through X.500 and related standard. With rfc2256 (1997) and rfc4519 (2006) the most important attributes and object classes were defined.

5.1.1 Standard Syntaxes and their usage

The most widespread syntax encodings:

Syntax Encodings	Description
Directory String	UTF-8 String. The most used Encoding
Generalized time	Very important encoding for data & time attributes. Allows comparison.
Numeric string	a sequence of digits
Boolean	TRUE/FALSE
Octet String	Binary value
Other Syntaxes	
DN	Distinguished Name.
Boolean	TRUE/FALSE
Telephone Number	very important and in widespread use
Postal Address	a number of strings separated by a dollar “$” sign.
Numeric string	a sequence of digits
IA5 String	a string of ASCII characters

5.1.2 The Basic Attribute types

The rfc2256 defines 55 standard Attributes. Rfc4524 and rfc4519 define together 68 attributes. A typical large implementation requires between 70 and 300 custom attributes. From this is clear that the authors of the standards didn’t wanted to define everything to the last detail but to give just a framework where the implementer could find room for a more detailed project.

The most used standard attributes are:

Name of the attribute	Description
objectClass	Needed for the every object for the object class definition.
cn	Common name: generic name of the object. For a person’s object it may be the complete name of given Name and family name.

sn	Surname. The family name of a person.
c	country: the ISO designation of a country: IT, DE, UK etc.
l	Locality.
street	Street of the address.
o	Organization's name. As this is often used as a container.
ou	Organisational Unit very often used as a container.
description	Free text.
postalAddress	Postal Address is the Address as printed on envelopes: a set of lines. The single lines are separated by a dollar sign.
postalCode	String containing the postal code.
telephoneNumber	Telephone number, normally coded including the international prefix.
facsimileTelephoneNumber	as telephoneNumber
member	Used for the object class "group", it contains only the DN of a member. The attribute is multivalued.
userPassword	Very special attribute!
givenName	Name of a Person.
uniqueMember	Multivalued attribute to represent members in a "groupOfUniqueNames".

5.2 Standard classes

Name of the object class	Description
person	the basic objectClass for person
residentialPerson	an extension for private persons
OrganizationalPerson	extended for persons in an organisation
inetOrgPerson	the most used objectClass to represent an user
groupOfNames	a collection of objects. widely used to assign a right.
groupOfUniqueNames	Sam as above with control about doublettes
organization	An object to represent an organisation, used mostly as a container for other objects.
organizationalUnit	As the objectClass "organization", in use to give a structure to a directory information tree. Seldom used to describe a division or a real organization.

1 Standard classes types

Name of the object	Description

class	
Abstract	a non-existent class, needed for logical completness. The most common ABSTRACT objectclass is top
Structural	can be used to create entries and can be part of a class hiearchy
Auxiliary	may be added into any convenient entry to add possible and mandatory attributes

6 Best Practices

- Remember that a schema is to be read and understood by many people beyond your department and you should not assume that they have direct access to written documentation.
- Spend enough time trying to find the right name for an attribute. Good names are an advantage for everybody while an obscure or a wrong name is a source of problems.
- If you have difficulties finding the right name for an attribute, please go back to the model, there are chances that something has not completely worked out in the design. It is normally very easy to find the right names in a good design.
-
- Use a short prefix for the attribute name this helps everybody to identify which attributes have been defined by you. In the examples the prefix “my” has been used. The custom attributes defined by my company Syntlogo have the prefix “sy”.
- Use an OID number, although this is going to be very important only if the project has a scope beyond a single organisation.
- Keep in mind that the attribute name is being written in the directory not once, but for every instance of the attribute itself, so length matters.
-
- Avoid names like: “xdTelephoneNumberPrimaryTieNumber”
- For complex, long and recurring concepts use an abbreviation and capitalize it: instead of “xdFacsimileTelephoneNumberCountryCode” you may use “xdFAXCountryCode”
- Avoid mixing languages in the definitions. Instead of “xdDepartmentODIKurzbezeichnung” name the attribute “xdODIDeptShortName”
- When using English to name objects, be grammatically correct: the container for all groups objects cannot be “ou=group” but has to be “ou=groupS”.
- Be careful not to be too generic: don’t call an Attribute “myStatus”: this is too generic, prefer instead “myEmploymentStatus”.
- On the other side, avoid overengineering your names.
- Define VERY WELL what you store in an attribute and HOW this will be stored. Format/conventions must apply throughout the Directory

- Do we need to fully describe an object or just make it searchable? Move some information to the connected applications, especially if the other partners are not interested in those details.

6.1 Information to Include in the Directory

What kind of information are we going to put into the directory? Any information about a person or asset can be added to an entry as an

attribute, but we should avoid storing information just for the sake of it: keep in mind that to have up-to-date and reliable information has normally a cost, so you should concentrate on the information that you really need.

For example:

- Contact information, such as telephone numbers, physical addresses, and email addresses.
- Descriptive information, such as an employee number, job title, manager or administrator identification, and job-related interests.
- Organization contact information, such as a telephone number, physical address, administrator identification, and business description.
- Contact and billing information for a corporation's trading partners, clients, and customers.
- Contract information, such as the customer's name, due dates, job description, and pricing information.
- Individual software preferences or software configuration information.
- Resource sites, such as pointers to web servers or the file system of a certain file or application.
- Contract or client account details
- Home contact information
- Office contact information for the various sites within the enterprise

6.2 Information to exclude

An LDAP Directory Server is excellent for managing large quantities of data that client applications read and write, but it is not designed to handle large, unstructured objects, such as images or other media.
These objects should be maintained in a file system. However, the directory can store pointers to these kinds of applications by using pointer URLs to FTP, HTTP, and other sites. In general, avoid storing blobs in the directory.
Once again, try to avoid information that is not needed by your business case: it risks being redundant and the cost of maintaining it is oft unnecessary.

6.3 How can we store Complex Data Structures?

How can we keep related Information together?
Let us imagine that we have to store a few postal addresses for an object. An address is made of the following information: 1) Street, 2) House number 3) Postal code 4) city 5) state 6) country.
We have the following possibilities:

a) The flat option: define all the attributes that you need and add a number to the definition.
   myAddress1Street

myAddress1StreetNumber
myAddress1PostalCode
myAddress1City
myAddress1State
myAddress1Country
myAddress2Street
myAddress2StreetNumber
myAddress2PostalCode
myAddress2City
myAddress2State
myAddress2Country
…..and so on….
Of course this option is viable only if the number of instances is VERY limited.

b) The container Option: Define a container object for one Address and append as many of them as needed to the main object.
Cumbersome and costly, but the best solution if the number of addresses is high.
You may even try innovative solutions if the addresses have oft duplicates: you may store them in a separate subtree and just reference them.

c) The structured Attribute Option: use an Attribute and define string inside it with separators or an XML object. This may not be searchable, but it solves the problem with a minor effort, if you don't have to share the solution with many partners.
This is used by the standard attribute "postalAddress", where a sequence of strings separated by a dollar sign represents the various lines of an address printed on an envelope.

7 The objects that we are going to use

We list here the most used building blocks our model. In a real design the will be a few more objects, but most of the time we restrict ourselves with this short list.

7.1 User

The user is the most important object of all the directories. Only by saying „user", we already said something: we are not interested in Persons, which have other attributes, like weight, eye colour, size and so on, but „users" a person who has a relationship with an organization and wants to access IT resources.
Disregarding most the qualities identifying a human being, we want just to:

1) identify the person in its interaction with the systems of the organization

2) its relationship with the organisation itself
3) and possibly something about the contracts and agreements with the same organisation which resolves into roles and access rights.

Typical information to identify a person:

a) Name, Family Name
b) Email, address
c) Telephone Number
d) Post address
e) Card ID, Social Security Number, Tax ID

Relationship with the Organisation

a) Personal number
b) Customer number
c) Organisational unit
d) Manager
e) Sales representative
f) Organisational role
g) Secretary
h) Cost centre

Contracts and agreements

a) Job description
b) Assigned task
c) Assigned Projects
d) Contract number
e) Exemptions
f) Entitlements
g) Allowances

7.2 Account

One model can be perfectly complete modelling only the „user“ object, knowing that the user has only one entry point to access the resources. There may be the case where a user may access the systems using different methods or having many entry point, not necessarily synchronized. In this case we must get the model more finely grained and introduce the „account“ object.

Typical information about an account

a) account identificator
b) password
c) system or related application
d) account privileges

It is responsibility of the architect to understand when a user object alone cover the needs of the system, and then to include the account's information in the user object itself, and when a separate object for the account is needed.

7.3 Organisational unit

An organisational Unit is very common in our models. Every organisation, even the smallest needs to clearly identify tasks and responsibilities. Even in our model such possibility to introduce a separation is welcome. The information conveyed by this type of container is limited, for our task to identify person and connect them to IT Ressources.
In early designs the organisation structure was rebuilt one to one using the objects „o" (organization) and „ou" (organizationalUnit), but this proved in short to be very inconvenient, as the changes of people inside an organisational units are frequent and the change of hierarchy on an object in a directory is a very expensive operation. So more recent designs abandoned the idea to accurately model the hierarchical structure and resort to store the organisational unit in an attribute of the person, very easily changed and put all users in just one container without any semantical meaning for the organisation.

7.4 Group

A group can be many things: a working group for a project, a group of friends: in most cases groups are a collection of users sharing some characteristic, often across organisational boundaries. Groups have often a group leader and some descriptive text to describe their scope.

In a directory, a group is a very convenient way to collect together a set of objects which has a characteristic in common.
We use oft this construct for many purposes but aside from the list of the members, we usually don't collect any further information: no group leader, no task description, although these may be added (you see, here begins the need to customise a schema).
So, if we need to have more information like the time when the member joined the group we have to store this somewhere else.
Of course, we may imagine a model where the group is a container and the objects inside it are the memberships themselves, and store all kinds of information in those objects, including the time when the member joined the group, who made the join and why.

8 The new Schema

8.1 How are we going to extend a standard class?

A good practice is to take a standard class and add the attributes that we need. To perform this we define an AUXILIARY CLASS, a class which is not structural, and add in the list of allowed attributes the new defined attributes.
This approach allows you to save most of the advantages of the standard and gives the possibility to add your additional information.

8.2 The user object

Let's start with the most central object of the design: the user object.
A good decision is to start with the inetOrgPerson, the starting point of almost all user objects. We now that we have to extend this class because we need some attributes that don't exist in the standards, like the Customer number.

The Class that we are going to define would be of the type auxiliary: this allows us to add needed information while keeping the convenience of a standard class as structural class, an important feature, if we happen later to have to use an application which only support standard classes.

To define an auxiliary class we must first define the attributes.

8.2.1 The „Customer Number"

The requirements ask for a customer number, and this attribute does not exist in the richest standard class the „inetOrgPerson". So we decide to define our first attribute.

a) Name: we are going to use our prefix „my" and the result is quite direct: myCustomerNumber.
b) Syntax: quite surprisingly, I am not suggesting a numeric string for the attribute type, but a „Directory String" with the „Case Ignore String" as matching rule. The reasons for that is fairly simple: first of all, we are not going to make any arithmetic operation on the customer number and secondly we don't want to put any artificial limit if later the "number" should be extended to symbols and letters. This choice is suggested every time you have to do with a numeric code.

8.2.2 The "delegate Administrator" information

We have many choices how to store this information: the simplest one is a Boolean attribute.

One step more complex is a “type”, a small string where we can store a User type and associate the value “01” to the delegated administrator.
If we make this choice, it is fairly straightforward to find the delegated administrator for a customer whose number is “123456”: you just use the following search (on more lines only for typographic clarity):

“(&(objectclass=myUser)
(myCustomerNumber=123456)
(myUserType=01))”

As you see, this idea leaves us much freedom to define further codes in the future.

We may have chosen to add this privilege as a general “role” attribute, where the entitlement as a delegated administrator could have been coded too, but this would have not given any advantage over this simple solution for very recurrent need to identify the administrator.

8.2.3 myId

We need to identify the object across many different applications and systems and the standard proposal to use „cn“ as relative distinguished name is not a good one: over the years the idea has that “cn” is “name + given name” and if we used this as a RDN we have a very large risk to have synonyms.
The best choice is to generate a unique code and use it as a RDN and call it „myID“.
This attribute is not designed to be seen or processed by human being, but it would be advisable to conceive in a simple fashion: it is going to be very useful for debugging, auditing, support, so use whatever gives you a few hexadecimal digits.
One last thing: being a unique identifier of the object, it is better that the attribute would be defined as „single Value“.

8.2.4 MyAlternateID

A very common practice is the use of an email as user identifier. We could be fairly sure that an email address does not have any collision in our database and we could even verify its correctness by sending an email to this address and asking the user to click the link contained in it. But we can be prepared for tomorrow and to the new incoming “identity providers”, where the users may have the strangest ID as they register coming from Facebook, Google or any future Vendor. For this reason we dedicate an attribute to this task “myAlterateIdentities”. Again Directory String, again Case Ignore String and absolutely MULTIVALUED.

8.2.5 Some standard attributes

givenName,sn, cn, mail, telephoneNumber

Why not? They are perfectly defined in purpose and gives us a reason to use a standard class.
Let's imagine for a while that a standard application needs to access the directory to accomplish its duties: for example a white pages search: having the most standard attributes filled is a guarantee that the application could still fulfil its scope.
Please note that the multivalued attribute „mail" will contain all mail alias while the mail address that has been verified previously is stored in myRegistredMail

8.2.6 Other custom attributes

myRegisteredMail stores the Mail address verified at the registration.
mySalutation is very requested in many countries like Germany.
MyBirthDate is important as a further check of identity, especially in countries where people can change family name (Germany).

8.2.7 Life Cycle Control

myDeactivationDate
myReductionDate

You are not going to delete user objects form a live directory and for this reason most directories are full of zombies. Of course it is a good idea to store the date when an account has been disabled (myDeactivationDate) and after six months to reduce it to a minimal form (myReductionDate), enough to use it in Auditing. After 10 years (number of years may depend from legislation) from the myReductionDate you may safely delete the object.
Please note that the syntax is „generalizedTime", a way to store year, month day, hours and minutes in reverse form („201511031230Z") and appending the zone information. This Syntax allows us to compare dates.

8.2.8 Password Control

myPasswordExpiryDate
myAccountControl

It is a cheap and good idea to store some flags into a „myAccountControl, as ActiveDirectory has taught. The numeric vale is converted in binary that the single bit has a specific meaning.
One of the flags requires that the password be changed at the next login.

It is sometimes difficult, without being in the hearth of the server, to know when your password would expire. And every server does it in a different way. A good way to avoid the problem its to store the expiry date explicitly in a generalized time attribute called „myPasswordExpiryDate“.

9 The Plumbing of the Schema

Once you get a clear Data Model you can define the characteristics of your custom attributes, your custom classes. Then you have to implement it: some LDAP Servers allow you to define the Schema extension with a GUI, others using text files but every of them stores the schema definition is a highly standardised object: the “cn=schema”. You can extend the schema adding new attributes and new classes, but you should not change anything if the standard is already present.

9.1.1 OID

An object identifier (OID) is a numeric string used to identify an object in LDAP. OIDs are used in schema, controls, and extended operations that require unique identification.

LDAP object classes and attributes require a base object identifier (OID) that must be unique to avoid naming conflicts.
If you plan to use your directory internally, you may use any OID but if you plan to export or publicly expose your schema in any way, you should consider entering a request for a unique OID.

After you have obtained a base OID, you can add branches to it for your organization's object classes and attributes. For Attributes you may append a .1 and the adding after that a counter. Then for classes you may append a .2 and so on.
For each component type, the directory server provides unique branch numbers to the base OID for each schema component.

For the examples we are using the Syntlogo GmbH Enterprise Number to build the OID: 1.3.6.1.4.1.13299.
The attributes will be
1.3.6.1.4.1.13299.1 and the counting
1.3.6.1.4.1.13299.1.2,
1.3.6.1.4.1.13299.1.3 and so on.

The object classes will receive and OID starting from 1.3.6.1.4.1.13299.2.1 and the counting.

To get you your own enterprise number, apply at the IANA:
http://pen.iana.org/pen/PenApplication.page

9.1.2 The general attribute definition

Following the Syntax definition given in the rfc and our attribute information collected previously, we are able now to code the input for the schema object as defined:

```
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-
Ord
["USAGE" whsp AttributeUsage ] ; default userApp. ext.
 whsp
")"
```

Examples:

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


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

1 myUser attributes list

Attribute	Syntax	Description
MUST		

cn	directory String, CIS	Name of the user: it will used for display it on dialogs and panels.
objectClass	directory String, CIS	organizationalPerson inetOrgPerson myUser
myID	**directory String, CIS**	**single valued**
ALLOW		
sn	directory String, CIS	Surname
givenName	directory String, CIS	Name
mail	rfc822 mailbox	
telephoneNumber	printable string	multivalued
mySalutation	**directory String, CIS**	
myBirthdate	**generalized time**	
myRegisteredMail	**rfc822mailbox**	**single valued**
myAlternateIdentity	**directory String, CIS**	**multivalued**
myUserType	**directory String, CIS**	
myDeactivationDate	**generalized time**	**multivalued**
myReductionDate	**generalized time**	**multivalued**
myAccountControl	**Binary/Numeric**	**bit Feld**
myCustomerNumber	**directory String, CIS**	**single valued**

Now we have all the components: the attributes definitions, the syntax, the OID. The generation of the attribute extension should be only a mechanical task:

```
attributeTypes: ( 1.3.6.1.4.1.13299.1.1 NAME ( 'myId' ) SYNTAX
1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.2 NAME ( 'mySalutation' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.3 NAME ( 'myBirthdate' ) SYNTAX
1.3.6.1.4.1.1466.115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.4 NAME ( 'myRegisteredMail' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.5 NAME ( 'myAlternateIdentity' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.6 NAME ( 'myUserType' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.7 NAME ( 'myDeactivationDate' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.8 NAME ( 'myReductionDate' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.9 NAME ( 'myAccountControl' )
SYNTAX 1.3.6.1.4.1.1466.115.121.1.6 )
attributeTypes: ( 1.3.6.1.4.1.13299.1.10 NAME
( 'myCustomerNumber' ) SYNTAX 1.3.6.1.4.1.1466.115.121.1.15  )
```

9.1.3 The general objectClass definition

Defintion in the rfc:

```
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 ")"
```

Example:

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

Our Class will be of the type “auxiliary”, to be able to add information without problems to an existing standard class:

```
objectClasses: (1.3.6.1.4.1.13299.2.1 NAME 'myUser'
SUP top
AUXILIARY
MAY ( myId $ mySalutation $ myBirthdate $ myRegisteredMail $
myAlternateIdentity $ myUserType $ myDeactivationDate $
myReductionDate $ myAccountControl $ myCustomerNumber )
)
```

A note about coding: you are not required to fold long lines as the above. But folding may help the readability.
Along the LDIF rfc, you may fold a line at any point, excluding the middle of an UTF-8 sequence, and insert a newline. The line where the content will be continues must be prepended by a blank. Keep in mind the blank will be discarded from the input.

```
attributeTypes: ( 1.3.6.1.4.1.13299.1.1 NAME ( 'myId' ) SYNTAX
1.3.6.1.4.1.1466.115.121.1.
 15 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.2 NAME ( 'mySalutation' ) SYNTAX
1.3.6.1.4.1.1466.11
 5.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.3 NAME ( 'myBirthdate' ) SYNTAX
1.3.6.1.4.1.1466.115
 .121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.4 NAME ( 'myRegisteredMail' ) SYNTAX
1.3.6.1.4.1.146
 6.115.121.1.26 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.5 NAME ( 'myAlternateIdentity' ) SYNTAX
1.3.6.1.4.1.
1466.115.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.6 NAME ( 'myUserType' ) SYNTAX
1.3.6.1.4.1.1466.115.
 121.1.15   )
attributeTypes: ( 1.3.6.1.4.1.13299.1.7 NAME ( 'myDeactivationDate' ) SYNTAX
1.3.6.1.4.1.1
 466.115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.8 NAME ( 'myReductionDate' ) SYNTAX
1.3.6.1.4.1.1466
 .115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.9 NAME ( 'myAccountControl' ) SYNTAX
1.3.6.1.4.1.146
 6.115.121.1.6 )
attributeTypes: ( 1.3.6.1.4.1.13299.1.10 NAME ( 'myCustomerNumber' ) SYNTAX
1.3.6.1.4.1.14
 66.115.121.1.15  )
objectClasses: (1.3.6.1.4.1.13299.2.1 NAME 'myUser' SUP top AUXILIARY MAY
( myId $ mySalut
 ation $ myBirthdate $ myRegisteredMail $ myAlternateIdentity $ myUserType $
myDeactivatio
  nDate $ myReductionDate $ myAccountControl $ myCustomerNumber ))
```

LDIF with line folding

```
dn cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( 1.3.6.1.4.1.13299.1.1 NAME ( 'myId' ) SYNTAX 1.3.6.1.4.1.1466.115.121.1.
 15 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.2 NAME ( 'mySalutation' ) SYNTAX 1.3.6.1.4.1.1466.11
 5.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.3 NAME ( 'myBirthdate' ) SYNTAX 1.3.6.1.4.1.1466.115
 .121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.4 NAME ( 'myRegisteredMail' ) SYNTAX 1.3.6.1.4.1.146
 6.115.121.1.26 SINGLE-VALUE  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.5 NAME ( 'myAlternateIdentity' ) SYNTAX 1.3.6.1.4.1.
1466.115.121.1.15  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.6 NAME ( 'myUserType' ) SYNTAX 1.3.6.1.4.1.1466.115.
 121.1.15   )
attributeTypes: ( 1.3.6.1.4.1.13299.1.7 NAME ( 'myDeactivationDate' ) SYNTAX 1.3.6.1.4.1.1
 466.115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.8 NAME ( 'myReductionDate' ) SYNTAX 1.3.6.1.4.1.1466
 .115.121.1.24  )
attributeTypes: ( 1.3.6.1.4.1.13299.1.9 NAME ( 'myAccountControl' ) SYNTAX 1.3.6.1.4.1.146
 6.115.121.1.6 )
attributeTypes: ( 1.3.6.1.4.1.13299.1.10 NAME ( 'myCustomerNumber' ) SYNTAX 1.3.6.1.4.1.14
 66.115.121.1.15  )
-
add: objectClasses
objectClasses: (1.3.6.1.4.1.13299.2.1 NAME 'myUser' SUP top AUXILIARY MAY ( myId $
mySalut
 ation $ myBirthdate $ myRegisteredMail $ myAlternateIdentity $ myUserType $ myDeactivatio
  nDate $ myReductionDate $ myAccountControl $ myCustomerNumber ))
-
```

10 Implement the design over the time

If you kept the design straightforward and simple, you are well prepared for the future: the capability to add an extra attribute or an object class is ALWAYS available.

It is going to be more difficult to remove an obsolete attribute. For once you have to make sure that no entries is using it and you might think to remove it from the schema definition, but then, how can you make sure that no application is requiring it?
The only reasonable action is to add the clause “OBSOLETE” to both the attribute and/or the class definition.
In this way you communicate to your colleagues, maintaining the directory years after your work that the attribute is not to be used anymore.

11 Conclusion

For many different reasons, like cost to gather and maintain accurate data, the cost of modifying and maintaining application accessing it, a successful directory is going to have a much longer life than most enterprise applications that it is serving.
As an example, I contributed to the design of a major directory deployment in 2001 storing information about 300.000 people and delivering information to 300+ applications. The directory is still alive in 2015, while most applications have been changed and despite the effort of the management to replace it with more modern and cheaper product.

12 References

During my work as a consultant and even during the conception of this tutorial I made use of a few sources of information that I warmly suggest for reference and further study.

URL to the source	Comment
http://www.openldap.org/doc/admin24/schema.html	The official reference on OpenLDAP, the Schema Specification section
http://www.zytrax.com/books/ldap/ch3/	A very interesting reference to LDAP and to OpenLDAP

https://www.rfc-editor.org/search/rfc_search_detail.php?page=All&title=LDAP&pubstatus[]=Standards%20Track&pubstatus[]=Best%20Current%20Practice&pubstatus[]=Informational&pubstatus[]=Experimental&std_trk=Any&pub_date_type=any&sortkey=Number&sorting=ASC	The List of LDAP RFC by IETF. For schema I use often rfc2256, rfc2307, rfc2798 (and of Course, rfc3698, 4510, 4524)
www.skills-1st.co.uk/papers/ldap-schema-design-feb-2005/ldap-schema-design-feb-2005.html	An interesting post from our host, Andrew Finlay
http://www.mitlinx.de/ldap/index.html?http://www.mitlinx.de/ldap/ldap_schema.htm	sorry: in German!
https://www.ldap.com/understanding-ldap-schema	from Unbound ID