TP N°3 LDAP

Master 2 I2L

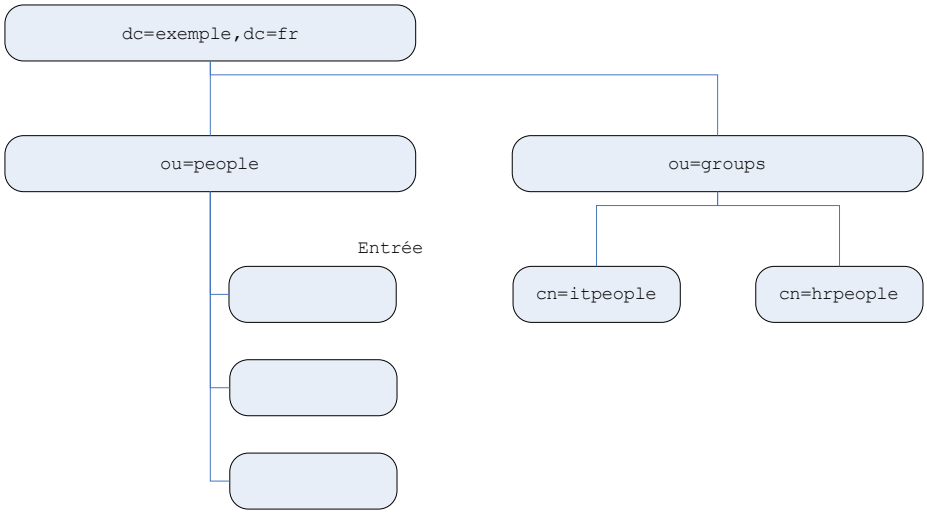
Année 2009/2010

D'après la version de Jean-Christophe Soulié (2007-2008)

D. Duvivier
LIL – Université du Littoral Côte d'Opale
duvivier@lil.univ-littoral.fr

1 Où on continue notre exemple…

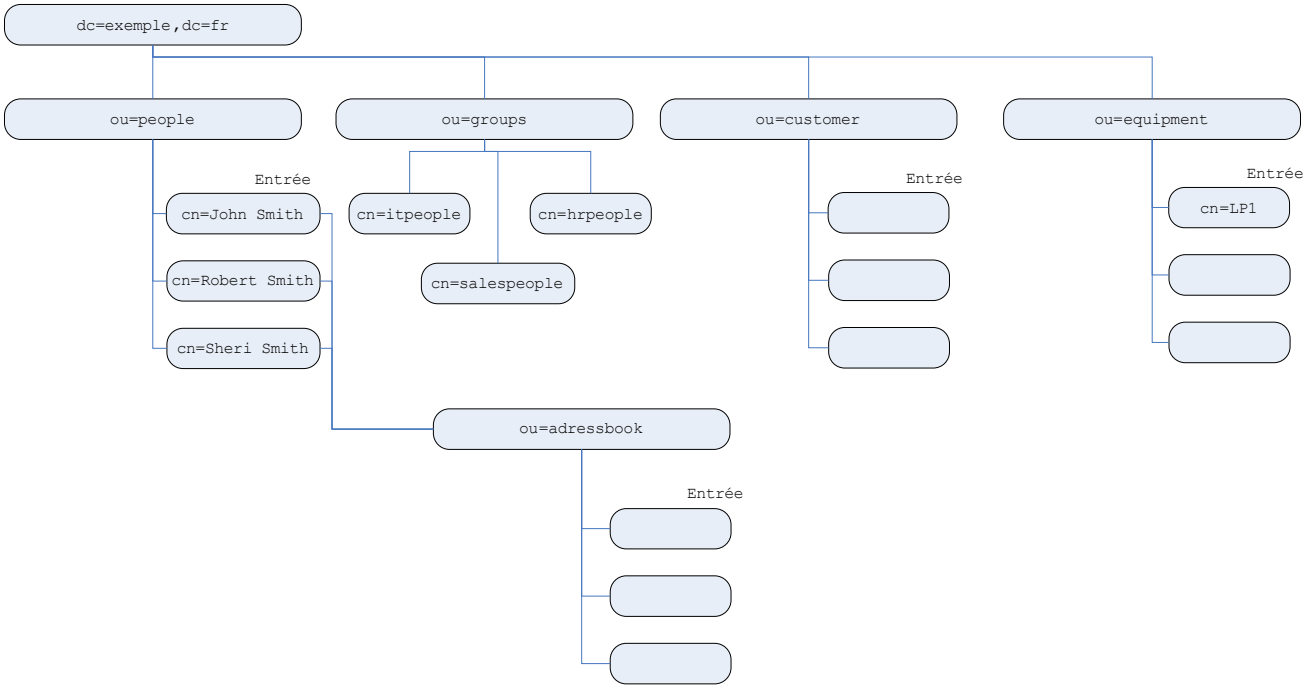
On reprend notre exemple précédent où nous avions l'arbre suivant et une politique d'accès liée au groupe (IT ou ressources humaines) :



Nous souhaitons améliorer cet arbre et apporter un certain nombre de modifications. Nous allons donc effectuer les ajouts suivants :

- Une base Client – Contact. Cette base pourra être lue par n'importe quel utilisateur authentifié, mais ne pourra avoir que des ajouts par le biais des ventes ;
- Tous les équipements informatiques (périphériques, PC, imprimantes, etc.) seront stockés dans une branche spéciale de l'arbre. Ces équipements pourront être mis à jour par les membres du groupe « `itpeople` » et lu par tout le monde ;
- Un carnet d'adresses personnel doit être ajouté dans l'arbre. Les entrées de ce carnet d'adresse sont relatives à une personne du groupe « `ou=people` ». Elles sont donc créées, lues et mise à jour seulement par la personne auquel est rattaché le carnet d'adresses. Mais la personne ne peut pas supprimer la branche du carnet d'adresses personnel (ce sont seulement les personne du groupe « `itpeople` »qui peuvent le faire) ;
- Les personnes du groupe « `hrpeople` » seront responsables d'ajouter et d'enlever des groupes.

Notre nouvel arbre aura donc l'aspect suivant :



Plus concrètement, on doit donc réaliser les opérations suivantes :

- Ajouter un nouveau groupe « salespeople » à notre branche « groups ». Sa description est : « Sales group » et John Smith est membre de ce groupe ;
- Ajouter une nouvelle branche « equipment » à notre branche principale. Cette branche sera du type objectclass prédéfini : « organizationalunit » ;
- Ajouter une nouvelle entrée à la branche ou=equipment. Le type d'objectclass prédéfini sera donc : « device ». Cette entrée aura les éléments suivants : cn=LP1, serialnumber=1-77-23-15, Description : ce que vous voulez !, localisation=Bureau 17. Elle est rattachée à John Smith et fait partie d'une unité d'organisation : « printer » ;
- Ajouter une nouvelle branche « customer » à notre branche principale. Cette branche sera du type objectclass prédéfini : « organizationalunit » ;
- Ajouter une nouvelle branche « addressbook » pour chaque personne. Cette branche sera du type objectclass prédéfini : « organizationalunit ».

Définir un fichier LDIF permettant de réaliser les opérations ci-dessus.

Définir les ACLs permettant de mettre en œuvre notre nouvelle politique de sécurité (Il y en a 10).

Vous pouvez maintenant tester vos ACLs de la manière suivante :

- Connectez-vous en tant que : cn=Robert Smith, ou=people, dc=example, dc=com. Comme cet utilisateur est membre du groupe hrpeople, il pourra voir et modifier toutes les entrées sauf les mots de passe (le sien bien sûr). De plus, il pourra voir les entrées customer et equipment, mais pas écrire. Enfin, il pourra ajouter, modifier des entrées dans son carnet d'adresses personnel et voir si les autres ont des carnets d'adresses, mais ne pas visualiser le contenu des autres carnets d'adresses.
- Connectez-vous en tant que : cn=Sheri Smith, ou=people, dc=example, dc=com. Comme cet utilisateur est membre du groupe itpeople, il pourra voir et modifier le mot de passe des autres utilisateurs, mais ne pas modifier les autres attributs (sauf les siens bien entendu !). Cet utilisateur peut lire les entrées customer, mais pas les modifier. Par contre, il peut voir et modifier les entrées equipment. Il peut aussi voir et modifier son propre carnet d'adresses. De plus, il peut voir si les autres ont des carnets d'adresses, mais ne pas visualiser le contenu des autres carnets d'adresses. Enfin, Il peut supprimer le carnet d'adresses des autres personnes.
- Connectez-vous en tant que : cn=John Smith, ou=people, dc=example, dc=com. Comme cet utilisateur n'a aucun privilège (ni hrpeople ou itpeople), il ne peut voir aucun des attributs : carlicense, homepostaladdress, homephone et userpassword des autres personnes mais les siens oui (qu'il peut modifier). Comme cet utilisateur est membre du groupe salespeople, il peut voir et modifier les entrées du groupe customer, mais peut seulement voir les entrées du groupe equipment. Il peut aussi voir et modifier son carnet d'adresses, mais ne pas voir celui des autres.
- Connectez-vous en tant qu'anonymous et vérifiez que vous ne pouvez pas vous connecter !
- Connectez-vous en tant qu'admin et vérifiez que vous pouvez (encore) tout faire !

2 Création et ajout d'objets

Dans cette partie, nous allons voir comment créer des attributs, un objectclass et un schéma pour pouvoir l'utiliser dans le futur.

Nous allons donc créer les attributs suivants dans la hiérarchie :

- L'attribut : dohicky. Cet attribut est une valeur booléenne. Elle peut seulement être vue et mise à jour par le propriétaire et par tous les membres du groupe hrpeople.
- L'attribut : ageAtBirth. Cet attribut est une valeur numérique. Elle peut seulement être vue et mise à jour par le propriétaire et par tous les membres du groupe hrpeople
- L'attribut : gobbledegook. Cet attribut est une chaîne de caractères et visible pour tout utilisateur authentifié. Par contre, il peut seulement être mis à jour par son propriétaire et par les membres du groupe hrpeople. Enfin, il va permettre de faire de recherche du type <= ou >=.

2.1 Mise en œuvre

A partir de la spécification précédente, nous allons suivre les étapes suivantes :

- L'attribut : dohicky. Nous allons créer le notre en utilisant la syntaxe OID boolean.
- L'attribut : ageAtBirth. Nous allons créer le notre en utilisant la syntaxe OID integer.
- L'attribut : gobbledegook. Nous allons créer le notre en utilisant la syntaxe OID PrintableString. Pour pouvoir utiliser les <= et >=, nous allons utiliser une ORDERING matching rule (caseIgnoreOrderingMatch).
- Nous allons créer un nouvel objectclass pour ces attributs que nous appellerons ourobject. Comme nous avons déjà un objet STRUCTURAL (inetorgperson), nous allons utiliser un AUXILIARY objectclass.
- Nous allons ensuite stocker notre travail dans un schéma que nous appellerons ourco.schema.

2.2 Un petit mot sur les OID

Les attributs vont suivre les conventions OID. Comme nous l'avons vu dans le cours, un OID (Object Identifier) est un nombre « globalement unique » défini par un organisme.

Avec LDAP, les OID sont utilisés pour identifier des objectclass, des attribute, des data type, des matchingrules, des protocol mechanisms, des controls, des extended operation et des supported features.

Un OID est un arbre structuré en une série de nombres qui sont séparés par des « . ». Par exemple :

```
2.5.6 # OID of x.500 objectclasses
2.5.6.2 # OID of country objectclass
1.3.6.1.4.1.1446 # Mark Whal (Critical Angle)
1.3.6.1.4.1.311 # microsoft's enterprise OID
1.2.840.113556 # microsoft's us OID
```

L'arbre OID est organisé par la gauche. C'est-à-dire que le nombre le plus à gauche est le plus haut niveau dans l'arbre et indique l'organisation internationale qui est responsable d'attribuer les nombres suivants dans la hiérarchie. La valeur la plus haute peut prendre les valeurs suivantes :

Number	Assignment
0	itu (IU-T : http://www.itu.int/)
1	iso (ISO : http://www.iso.org/)
2	joint itu-iso

La base OID 2.5 a été assigné par itu-iso au groupe d'étude X.500. Par conséquent, les nombres commençant par 2.5 (2.5.6.x ou 2.5.4.x par exemple) sont attribué par le groupe d'étude X.500.
La base OID 1.3.6.1.4.1 est utilisée pour les séquences privées des entreprises qui sont définies par IANA (http://www.iana.org/protocols/).

Exemple :

```
# X below is the enterprise number assigned by IANA
1.3.6.1.4.1.X.1 - assign to SNMP objects
1.3.6.1.4.1.X.2 - assign to LDAP objects
1.3.6.1.4.1.X.2.1 - assign to LDAP syntaxes
1.3.6.1.4.1.X.2.2 - assign to LDAP matchingrules
1.3.6.1.4.1.X.2.3 - assign to LDAP attributes
1.3.6.1.4.1.X.2.4 - assign to LDAP objectclasses
1.3.6.1.4.1.X.2.5 - assign to LDAP supported features
1.3.6.1.4.1.X.2.9 - assign to LDAP protocol mechanisms
1.3.6.1.4.1.X.2.10 - assign to LDAP controls
1.3.6.1.4.1.X.2.11 - assign to LDAP extended operations
```

Syntaxe Booléenne :

```
Boolean (OID=1.3.6.1.4.1.1466.115.121.1.7) Permet une matchingRule de booleanMatch
```

Syntaxe Numérique :
Les valeurs numériques sont stockées avec des chaînes de caractères, mais elles permettent des comparaisons numérique, par exemple : 100 > 2.

```
Integer (OID=1.3.6.1.4.1.1466.115.121.1.27) Permet une matchingRules de integerMatch et integerOrderingMatch
```

Syntaxe PrintableString :

```
PrintableString (OID=1.3.6.1.4.1.1466.115.121.1.44) chaîne de caractères IA5 (almost ASCII) mais limitée aux caractères définis dans la RFC 2252 section 4.1 production p. Ne permet pas d'utiliser des caractères étendus tels que : é, Ø, å etc. Permet une matchingRules de caseIgnoreMatch et caseIgnoreSubstringsMatch.
```

2.3 Définitions des attributs

dohicky – boolean

```
attributetype ( 1.3.6.1.4.1.6863.2.3.107 NAME 'dohicky'
     EQUALITY booleanMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
```

ageAtBirth – integer

```
attributetype ( 1.3.6.1.4.1.6863.2.3.108 NAME 'ageAtBirth'
     EQUALITY integerMatch
     ORDERING integerOrderingMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
```

gobbledegook – PrintableString

```
attributetype ( 1.3.6.1.4.1.6863.2.3.109 NAME 'gobbledegook'
     EQUALITY caseIgnoreMatch
     SUBSTR caseIgnoreSubstringsMatch
     ORDERING caseIgnoreOrderingMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.44{200} ) (200 caractères max pour cet attribut)
```

2.4 Objectclass et Définition du schéma

```
objectclass ( 1.3.6.1.4.1.6863.2.4.57 NAME 'ourObject'
     DESC 'A very useful object'
     SUP top AUXILIARY
     MUST ( dohicky $ gobbledegook )
     MAY ageAtBirth )
```

- `SUP top` termine la hiérarchie objectclass en utilisant les caractéristiques de l'objet `top`.
- `AUXILIARY` permet d'associer cet objectclass avec n'importe quel objectclass de type STRUCTURAL.
- `MUST ( dohicky $ gobbledegook )` définit que les attributs `dohicky` et `gobbledegook` doivent être présent quand l'objectclass est ajouté.
- `MAY ageAtBirth` indique que cet attribut est optionnel.

Nous pouvons maintenant définir complètement notre fichier `ourco.schema` :

```
attributetype ( 1.3.6.1.4.1.6863.2.3.107 NAME 'dohicky'
     EQUALITY booleanMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )

attributetype ( 1.3.6.1.4.1.6863.2.3.108 NAME 'ageAtBirth'
     EQUALITY integerMatch
     ORDERING integerOrderingMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )

attributetype ( 1.3.6.1.4.1.6863.2.3.109 NAME 'gobbledegook'
     EQUALITY caseIgnoreMatch
     SUBSTR caseIgnoreSubstringsMatch
     ORDERING caseIgnoreOrderingMatch
     SYNTAX 1.3.6.1.4.1.1466.115.121.1.44{200} )

objectclass ( 1.3.6.1.4.1.6863.2.4.57 NAME 'ourObject'
     DESC 'A very useful object'
     SUP top AUXILIARY
     MUST ( dohicky $ gobbledegook )
     MAY ageAtBirth )
```

2.5 Modification du fichier slapd.conf

Il faut juste ajouter la ligne suivante (au bon endroit !) pour ajouter notre nouveau schéma :

```
include /path/vers/votre/fichier/schema/ourco.schema
```

Et rajouter l'ACL qui permet de prendre en compte notre politique de sécurité pour ces attributs.

```
access to attr=homedirectory,uidnumber,gidnumber,loginshell,gecos
     by group.exact="cn=itpeople,ou=groups,dc=exemple,dc=fr" write
     by *          none
```

Et il faut aussi modifier une autre ACL.

2.6 Modification des entrées

On commence par enlever toutes les entrées existantes (fichier `delete_all.ldif`) :

```
dn: ou=addressbook,cn=John Smith,ou=people,dc=exemple,dc=fr
changetype: delete

dn: cn=John Smith,ou=people,dc=exemple,dc=fr
changetype: delete

dn: ou=addressbook,cn=Robert Smith,ou=people,dc=exemple,dc=fr
changetype: delete

dn: cn=Robert Smith,ou=people,dc=exemple,dc=fr
changetype: delete

dn: ou=addressbook,cn=Sheri Smith,ou=people,dc=exemple,dc=fr
changetype: delete

dn: cn=Sheri Smith,ou=people,dc=exemple,dc=fr
changetype: delete
```

Et avec la commande :

```
ldapmodify -v -H ldap://localhost -x -D "cn=admin,dc=exemple,dc=fr" -f
delete_all.ldif -w secret
```

On enlève tout ce qui concerne les personnes de la partie `people`.

Enfin, on repeuple cette partie de la base (avec le fichier `ajout.ldif`) :

```
dn: cn=John Smith,ou=people,dc=exemple,dc=fr
changetype: add
objectclass: inetorgperson
carLicense: HISCAR 124
cn: John Smith
cn: John J Smith
homePhone: 555-111-2223
mail: j.smith@example.com
mail: jsmith@example.com
mail: john.smith@example.com
ou: Sales
sn: Smith
uid: jsmith
userPassword: jSmitH
objectclass: posixaccount
uidnumber: 200
gidnumber: 203
homedirectory: /var/mail/exemple.fr/jsmith
objectclass: ourObject
dohicky: FALSE
gobbledegook: john
ageatbirth: 0

dn: ou=addressbook,cn=John Smith,ou=people,dc=exemple,dc=fr
changetype: add
description: Personal Address Book
objectClass: organizationalUnit
ou: addressbook

dn: cn=Robert Smith,ou=people,dc=exemple,dc=fr
changetype: add
objectclass: inetorgperson
carLicense: HISCAR 123
cn: Robert Smith
cn: Bob J Smith
homePhone: 555-111-2222
mail: robert.smith@example.com
mail: bob.smith@example.com
ou: Human Resources
sn: smith
telephoneNumber: 555-555-1212
telephoneNumber: 212
title: Department Manager
uid: rjosmith
userPassword: rJsmitH
objectclass: posixaccount
uidnumber: 200
gidnumber: 203
homedirectory: /var/mail/exemple.fr/ssmith
objectclass: ourObject
dohicky: TRUE
gobbledegook: sheri

dn: ou=addressbook,cn=Robert Smith,ou=people,dc=exemple,dc=fr
changetype: add
description: Personal Address Book
objectClass: organizationalUnit
ou: addressbook

dn: cn=Sheri Smith,ou=people,dc=exemple,dc=fr
changetype: add
```

```
objectclass: inetorgperson
carLicense: HERCAR 125
cn: Sheri Smith
homePhone: 555-111-2225
mail: s.smith@example.com
mail: ssmith@example.com
mail: sheri.smith@example.com
ou: IT
sn: smith
uid: ssmith
userPassword: sSmitH
objectclass: posixaccount
uidnumber: 200
gidnumber: 203
homedirectory: /var/mail/exemple.fr/ssmith
objectclass: ourObject
dohicky: FALSE
gobbledegook: robert
ageatbirth: 17

dn: ou=addressbook,cn=Sheri Smith,ou=people,dc=exemple,dc=fr
changetype: add
description: Personal Address Book
objectClass: organizationalUnit
ou: addressbook
```

Et avec la commande :

```
ldapmodify -v -H ldap://localhost -x -D "cn=admin,dc=exemple,dc=fr" -f ajout.ldif
-w secret
```

On ajoute tout ce qui concerne les personnes de la partie `people`.

2.7 Tests !

- Connectez-vous en tant que : `cn=Robert Smith, ou=people, dc=example, dc=com`. Comme cet utilisateur est membre du groupe `hrpeople`, il pourra voir et modifier toutes les entrées sauf les mots de passe (le sien bien sûr). De plus, il pourra voir les entrées `customer` et `equipment`, mais pas écrire. Enfin, il pourra ajouter, modifier des entrées dans son carnet d'adresses personnel et voir si les autres ont des carnets d'adresses, mais ne pas visualiser le contenu des autres carnets d'adresses. Il ne peut pas voir les attributs de l'object `posixaccount` (`homedirectory`, `guidnumber`, `gidnumber`, `loginshell` et `gecos`), même pour le sien et aussi pour les autres.
- Connectez-vous en tant que : `cn=Sheri Smith, ou=people, dc=example, dc=com`. Comme cet utilisateur est membre du groupe `itpeople`, il pourra voir et modifier le mot de passe des autres utilisateurs et les attributs : `homedirectory`, `guidnumber`, `gidnumber`, `loginshell` et `gecos` ; mais ne pas modifier les autres attributs (sauf les siens bien entendu !). Cet utilisateur peut lire les entrées `customer`, mais pas les modifier. Par contre, il peut voir et modifier les entrées `equipment`. Il peut aussi voir et modifier son propre carnet d'adresses. De plus, il peut voir si les autres ont des carnets d'adresses, mais ne pas visualiser le contenu des autres carnets d'adresses. Enfin, Il peut supprimer le carnet d'adresses des autres personnes.
- Connectez-vous en tant que : cn=John Smith, ou=people, dc=example, dc=com. Comme cet utilisateur n'a aucun privilège (ni `hrpeople` ou `itpeople`), il ne peut voir aucun des attributs : `carlicense`, `homepostaladdress`, `homephone` et `userpassword` des autres personnes mais les siens oui (qu'il peut modifier). Comme cet utilisateur est membre du groupe `salespeople`, il peut voir et modifier les entrées du groupe `customer`, mais peut seulement voir les entrées du groupe `equipment`. Il peut aussi voir et modifier son carnet d'adresses, mais ne pas voir celui des autres. Il ne peut pas voir les attributs de l'object `posixaccount` (`homedirectory`, `guidnumber`, `gidnumber`, `loginshell` et `gecos`), même pour le sien et aussi pour les autres.
- Connectez-vous en tant qu'`anonymous` et vérifiez que vous ne pouvez pas vous connecter !
- Connectez-vous en tant qu'`admin` et vérifiez que vous pouvez (encore) tout faire !

3 Annexe sur les ACL

Format:

```
access to <what> [ by <who> <accesslevel> <control> ]+
```

A set of **access** statements create what is sometimes known as **Access Control Lists (ACL)** or **Access Control Policies (ACP)**.

The **access** directive may be placed in either the **global** or the **database** section of the slapd.conf file and grants access (specified by <accesslevel>) to a set of entries and/or attributes (specified by <what>) by one or more requesters (specified by <who>). Multiple **access** directives may be included.

The access directive (ACL) is brutally complex. It allows very fine control over who can access what objects and attributes and under what conditions. The side-effect of this complexity and power is that it is very easy to get the **access** directive wrong. You must thoroughly test ACL directives with all possible permissions. slapacl is an automated tool to test access to specific attributes and entries based on the current access directives.

to <what>

The entity the access control directive applies to. Multiple what entries can be included in a single directive. This statement can have the following forms:

*	A wildcard that stands for all the entries.
dn[.**dnstyle**]=**pattern**	This form defines an entry based on its DN (a quoted string) e.g. dn="ou=people,dc=example,dc=com". **dnstyle** is an optional qualifier which may take one of the following values:.

regex	Regex is defaulted if **dnstyle** is omitted. If a regular expression is enclosed in parenthesis () the resulting substrings (**submatches**) may be used in following <who> clauses using the form $**digit**, with digit ranging from 1 to 9 and $1 being substituted from the first submatch $2 from the second and so on. Example:

```
# assume supplied dn
# is ou=something,cn=my name,dc=example,dc=com
# then $1 = my name at end of match below
# because first expression does not have ()
access to dn="ou=[^,]+,cn=([^,]+),dc=example,dc=com"
 by dn.exact="cn=$1,dc=example,dc=com"
```

base	(exact is an alias) only the entry addressed by **pattern**
one	indicates all entries immediately below **pattern**
subtree	indicates all the subentries (lower levels) under **pattern** but including **pattern**.
children	indicates all the subentries (lower levels) under **pattern** but excluding **pattern**.

attrs=**attrlist**	A single or comma separated list of attributes to which this access control directive applies. (Note: accepted either **attr** or **attrs** you will see both in the doumentation with no explanation. Current (2.4) is now warning that attr is deprecated so parameter changed to **attrs** in docs and all sample files). There are three additional **pseudo-attributes** that may be used:

entry	scope limited to this entry
children	allows access to the child objects of the entry identified
@objectclass	**OpenLDAP 2.2+** the **@** implies all attributes of the objectclass

filter=**ldapfilters**	String representation of a search filter.

Examples:

```
# NOTE: These are snippets only - the dots (...)
#       indicate that there may be more data
#       the dots should not be present on final directive

# access to the defined attributes
access to attr=userpassword,honephone ...

# access to the defined DN and all lower levels
# dnstyle is missing so regex is defaulted
# covers all DNs below the defined DN as well
access to dn="ou=people,dc=example,dc=com" ...

# access to one level below the defined DN only
access to dn.one="ou=people,dc=example,dc=com" ...

# access to one level below the defined DN only
# for attribute userpassword only
access to dn.one="ou=people,dc=example,dc=com"
    attr=userpassword ...
```

by <who>

Multiple <who> clauses can appear in an access control statement. It can take the following forms:

*	A wildcard that stands for everyone.
anonymous	Access is granted to unauthenticated users. May be used in conjunction with **auth** e.g. `... by anonymous auth` Allows OpenLDAP to access an authentication attribute on behalf of an anonymous user solely for the purposes of authenticating that user.
users	Grants access to authenticated users.
self	Access to an entry is allowed if the entry authenticates with a password used in that entry.
dn[.**dnstyle**[,**modifier**]]=**pattern**	Access is granted to the matching DN. The optional **dnstyle** allows the same choices as for the dn form of the **what** field. **pattern** is a quoted string. The keyword **expand** may be used in conjunction with **dnstyle** this indicates that **submatch** substitition should be performed on value of the form **$<digit>**. **Note:** We note that version 2.4 will frequently (but not always) reject the use of the modifier **expand** suggesting that it is implied (when used with **regex**) and therefore not necessary. Example: `access to dn.regex="^cn=([^,]+),ou=People,dc=example,dc=com$" # assume cn=my entry,ou=People,dc=example,dc=com # $1 has the value 'my entry' which is substituted below by dn.exact,expand="cn=$1,ou=People,dc=example,dc=com" read`
dnattr=**attrname**	Access is granted to requests whose DN is listed in the entry being accessed under the attrname attribute.
group[/**objectclass**[/**attrname**]][.style]=**pattern**	Access is granted to the group entry whose DN is given by **pattern**. The optional parameters **objectclass** and **attrname** may be used to qualify the attribute used to determinine membership (**member** is assumed by default). The optional **style** can be **regex** (default if missing), or **exact** (base is an alias) - exact match only. **pattern** is a quoted string.
peername[.**style**]=**pattern** sockname[.**style**]=**pattern** sockurl[.**style**]=**pattern**	The source IP (**peername** form), the named pipe file name (**sockname** form) or the source URL (**sockurl** form) are compared against **pattern** to determine access. The optional **style** can be **regex** (default if missing), or **exact** (base is an alias) - exact match only.
domain[.**style**[,**modifier**]]=**pattern**	The source host name is compared against **pattern** to determine access. The optional **style** can be **regex** (default if missing), or **exact** (base is an alias) - exact match only or **subtree** which matches when a fully qualified name exactly matches the domain pattern, or its trailing part, after a dot, exactly matches the domain pattern. The domain of the source host is obtained by performing a DNS reverse lookup using the source IP. As this lookup can easily be spoofed, use of the domain statement is strongly discouraged. By default, reverse lookups are disabled (see reverse lookup).
set[.**style**]=**pattern**	to be supplied
ssf=**n** transport_ssf=**n** tls_ssf=**n** sasl_ssf=**n**	Sets the required Security Strength Factor (ssf) required to grant access.
aci=**attrname**	Access control is determined by the values in the attrname of the entry itself. ACIs are experimental; they must be enabled at compile time.

<accesslevel>

accesslevel determines the access level or the specific access privileges the who field will have and takes the following form.

```
[self]{<level>|<priv>}
```

Parameters are:

self The modifier self allows special operations like having a certain access level or privilege only in case the operation involves the name of the user that's requesting the access. It implies the user that requests access is bound. An example is the selfwrite access to the member attribute of a group, which allows one to add/delete its own DN from the member list of a group, without affecting other members.

level The level access model relies on an incremental interpretation of the access privileges. The possible levels are defined below. Each access level implies all the preceding ones, thus write access will imply all accesses, search allows search, compare and auth.

May take one of the following values:

none	No access is allowed.
auth	Means that OpenLDAP is allowed internal access to the attributes defined in the <what> clause for the purposes of authentication/authorization (e.g. bind) only.
compare	
search	The attributes defined in the <what> clause are allowed to be accessed for searching.
read	The attributes defined in the <what> clause may be read.
write	The attributes defined in the <what> clause may be written to.

priv The priv access model relies on the explicit setting of access privileges for each clause. The = sign resets previously defined accesses; as a consequence, the final access privileges will be only those defined by the clause. The + and - signs add/remove access privileges to the existing ones. The privileges are w for write, r for read, s for search, c for compare, and x for authentication. More than one privilege can be added in one statement.

has the format:

```
{=|+|-}{w|r|s|c|x}+
```

<control>

control is optional and if present takes one of the following values

stop	The default, means access checking stops in case of match.
continue	the continue form allows for other <who> clauses in the same <access> clause to be considered, so that they may result in incrementally altering the privileges,
break	Allows for other <access> clauses that match the same target to be processed.

And that is all there is to the **access** directive. Pretty simple really!

4 Annexe sur les expressions régulières

4.1 Some Definitions before we start

We are going to be using the terms **literal**, **metacharacter**, **target string**, **escape sequence** and **search string** in this overview.

Here is a definition of our terms:

literal — A **literal** is any character we use in a search or matching expression, for example, to find **ind** in w**ind**ows the **ind** is a **literal** string - each character plays a part in the search, it is **literally** the string we want to find.

metacharacter — A **metacharacter** is one or more special characters that have a unique meaning and are NOT used as **literals** in the search expression, for example, the character ^ (circumflex or caret) is a **metacharacter**.

escape sequence — An **escape sequence** is a way of indicating that we want to use one of our **metacharacters** as a **literal**. In a regular expression an **escape sequence** involves placing the **metacharacter** \ (backslash) in front of the **metacharacter** that we want to use as a **literal**, for example, if we want to find **^ind** in **w^indow** then we use the search string **\^ind** and if we want to find **\\file** in the string **c:\\file** then we would need to use the search string **\\\\file** (each \ we want to search for (a **literal**) is preceded by an **escape sequence** \).

target string — We have chosen to use this term to describe the string that we will be searching, that is, the string in which we want to find our match or search pattern.

search expression — We have chosen to use this term to describe the expression that we will be using to search our target string, that is, the pattern we use to find what we want.

4.2 Our Example Target Strings

Throughout this guide we will use the following as our target strings:

```
STRING1   Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)
STRING2   Mozilla/4.75 [en](X11;U;Linux2.2.16-22 i586)
```

4.2.1 Simple Matching

We are going to try some simple matching against our example target strings:

4.2.1.1 Search for

m	**STRING1**	match	Finds the **m** in co**m**patible
	STRING2	no match	There is no lower case **m** in this string. Searches are case sensitive unless you take special action.
a/4	**STRING1**	match	Found in Mozill**a/4**.0 - any combination of characters can be used for the match
	STRING2	match	Found in same place as in STRING1
5 [	**STRING1**	no match	The search is looking for a pattern of '5 [' and this does NOT exist in STRING1. Spaces are valid in searches.
	STRING2	match	Found in Mozilla/4.7**5 [**en]
in	**STRING1**	match	found in W**in**dows
	STRING2	match	Found in L**in**ux
le	**STRING1**	match	found in compatib**le**
	STRING2	no match	There is an l and an e in this string but they are not adjacent (or contiguous).

4.2.2 Brackets, Ranges and Negation

Bracket expressions introduce our first **metacharacters**, in this case the square brackets which allow us to define list of things to test for rather than the single characters we have been checking up until now. These lists can be grouped into what are known as Character Classes typically comprising well know groups such as all numbers etc.

4.2.2.1 Metacharacter *Meaning*

[ ]	Match anything inside the square brackets for one character position once and only once, for example, [12] means match the target to either 1 or 2 while [0123456789] means match to any character in the range 0 to 9.
-	The - (dash) **inside square brackets** is the 'range separator' and allows us to define a range, in our example above of [0123456789] we could rewrite it as [0-9]. You can define more than one range inside a list e.g. [0-9A-C] means check for 0 to 9 and A to C (but not a to c). **NOTE:** To test for - inside brackets (as a **literal**) it must come first or last, that is, [-0-9] will test for - and 0 to 9.
^	The ^ (circumflex or caret) **inside square brackets** negates the expression (we will see an alternate use for the circumflex/caret **outside** square brackets later), for example, [^Ff] means anything except upper or lower case F and [^a-z] means everything except lower case a to z. **NOTE:** Spaces, or in this case the lack of them, between ranges are very important.

NOTE: There are some special range values (Character Classes) that are built-in to most regular expression software and have to be if it claims POSIX 1003.2 compliance for either BRE or ERE.

So lets try this new stuff with our target strings.

4.2.2.2 Search for

in[du]	**STRING1**	match	finds ind in W**ind**ows
	STRING2	match	finds **inu** in L**inu**x
x[0-9A-Z]	**STRING1**	no match	Again the tests are case sensitive to find the xt in DigE**xt** we would need to use [0-9a-z] or [0-9A-Zt]. We can also use this format for testing upper and lower case e.g. [Ff] will check for lower and upper case F.
	STRING2	match	Finds x2 in Linu**x2**
[^A-M]in	**STRING1**	match	Finds Win in **Win**dows
	STRING2	no match	We have excluded the range A to M in our search so Linux is not found but linux (if it were present) would be found.

4.2.3 Positioning (or Anchors)

We can control where in our target strings the matches are valid. The following is a list of **metacharacters** that affect the position of the search:

4.2.3.1 Metacharacter *Meaning*

^	The ^ (circumflex or caret) **outside square brackets** means look only at the beginning of the target string, for example, **^Win** will not find Windows in **STRING1** but **^Moz** will find **Moz**illa.
$	The $ (dollar) means look only at the end of the target string, for example, fox$ (can be written as $fox) will find a match in 'silver **fox**' but not in 'the fox jumped over the moon'.
.	The . (period) means any character(s) in this position, for example, **ton.** will find **tons** and **tonneau** but not **wanton** because it has no following character.

NOTE: Many systems and utilities, but not all, support special positioning macros, for example \< match at beginning of word, \> match at end of word, \b match at the begining OR end of word , \B except at the beginning or end of a word. So let's try this lot out with our example target strings...

4.2.3.2 Search for

$[a-z])	**STRING1**	match	finds t) in DigiEx**t)**
	STRING2	no match	We have a numeric value at the end of this string but we would need [0-9a-z]) to find it.
.in	**STRING1**	match	Finds **Win** in **Win**dows.
	STRING2	match	Finds Lin in **Lin**ux.

4.2.4 Iteration 'metacharacters'

The following is a set of **iteration metacharacters** (a.k.a. quantifiers) that can control the number of times a character or string is found in our searches.

4.2.4.1 Metacharacter *Meaning*

?	The ? (question mark) matches the preceding character 0 or 1 times only, for example, colou?r will find both color and colour.
*	The * (asterisk or star) matches the preceding character 0 or more times, for example, tre* will find tree and tread and trough.
+	The + (plus) matches the previous character 1 or more times, for example, tre+ will find tree and tread but not trough.
{n}	Matches the preceding character n times exactly, for example, to find a local phone number we could use [0-9]{3}-[0-9]{4} which would find any number of the form 123-4567. **Note:** The - (dash) in this case, because it is outside the square brackets, is a **literal**. Value is enclosed in braces (curly brackets).
{n,m}	Matches the preceding character at least n times but not more than m times, for example, 'ba{2,3}b' will find 'baab' and 'baaab' but NOT 'bab' or 'baaaab'. Values are enclosed in braces (curly brackets).

So let's try them out with our example target strings.

4.2.4.2 Search for

\(.*l	**STRING1**	match	finds l in compatib**l**e (**Note:** The opening \ is an escape sequence used to indicate the (it precedes is a literal not a metacharacter.)
	STRING2	no match	Mozilla contains lls but not preceded by an open parenthesis (no match) and Linux has an upper case L (no match).

We had previously defined the above test using the search value **l?** (thanks to David Werner Wiebe for pointing out our error). The search expression l? actually means find anything, even if it has no l (l 0 or 1 times), so would match on both strings. We had been looking for a method to find a single l and exclude ll which, without lookahead (a relatively new extension to regular expressions pioneered by PERL) is pretty difficult. Well that is our excuse.

W*in	**STRING1**	match	Finds the Win in **Win**dows.
	STRING2	match	Finds **in** in L**in**ux preceded by W zero times - so a match.
[xX][0-9a-z]{2}	**STRING1**	no match	Finds x in DigExt but only one t.
	STRING2	match	Finds X and 11 in X11.

4.2.5 More 'metacharacters'

The following is a set of additional **metacharacters** that provide added power to our searches:

4.2.5.1 Metacharacter	*Meaning*
()	The ((open parenthesis) and) (close parenthesis) may be used to group (or bind) parts of our search expression together.
|	The | (vertical bar or pipe) is called **alternation** in techspeak and means find the left hand OR right values, for example, gr(a|e)y will find 'gray' or 'grey'.

So lets try these out with our example strings...

4.2.5.2 Search for			
^([M-Z]in)	**STRING1**	no match	The '^' is an anchor indicating first position. **Win** does not start the string so no match.
	STRING2	no match	The '^' is an anchor indicating first position. **Lin**ux does not start the string so no match.
((4\.[0-3])|(2\.[0-3]))	**STRING1**	match	Finds the **4.0** in Mozilla/**4.0**.
	STRING2	match	Finds the **2.2** in Linux**2.2**.16-22.
(W|L)in	**STRING1**	match	Finds **Win** in **Win**dows.
	STRING2	match	Finds **Lin** in **Lin**ux.