Moulinette Documentation

Release 2.6.1

YunoHost Collective

Jul 27, 2020

Contents:

https://github.com/yunohost/Moulinette is the internal framework used by YunoHost to generate both its web Rest API and CLI interface.

This framework is aimed at: define once, get both CLI and web Rest API at the same time, offering easy and fast development and a ubiquitous experience from different interfaces.

This documentation is mainly aimed for the YunoHost's collective and serves as a reference.

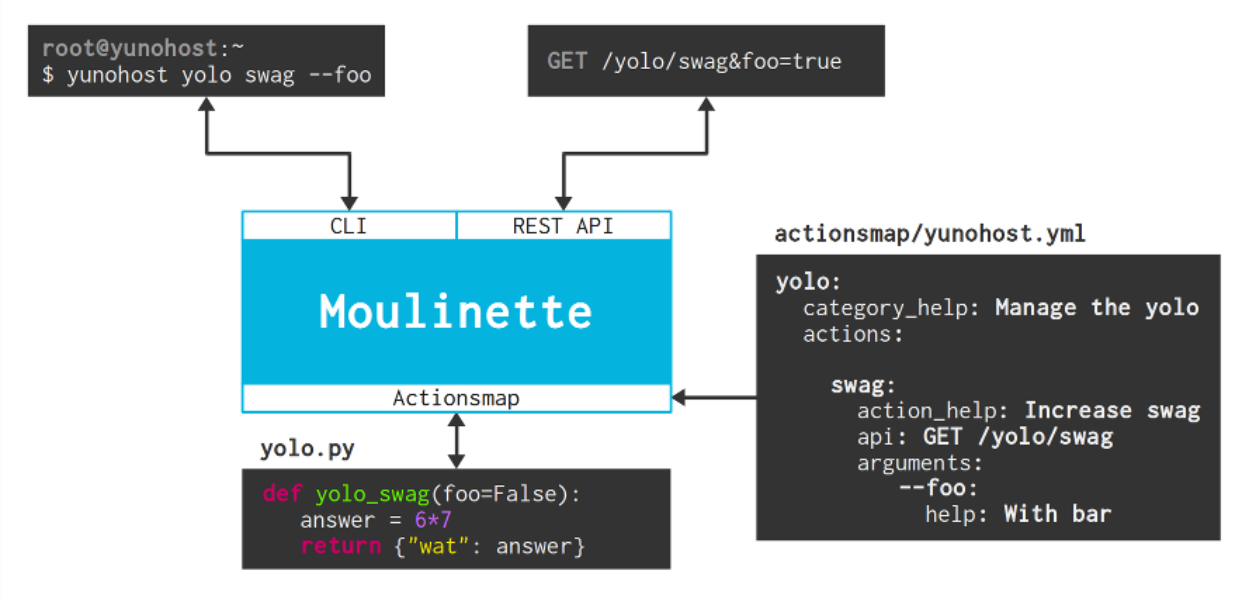
CHAPTER 1

Role and syntax of the actionsmap

1.1 Principle

The actionsmap allows to easily define commands and their arguments through a YAML file. Moulinette will automatically make the command available through the CLI and Rest API, and will be mapped to a python function.

The illustration below summarizes how it works :



1.2 Format of the actionmap

General description of categories/subcategories, actions, arguments.

1.2.1 Authentication configuration

Document the *configuration: authenticate: all* LDAP stuff ...

1.2.2 Special options for arguments

Document *nargs*, *metavar*, *extra: pattern*

CHAPTER 2

Common LDAP operation (for YunoHost but not only)

Moulinette is deeply integrated with LDAP which is used for a series of things like:

- storing users
- storing domains (for users emails)
- SSO

This page document how to uses it on a programming side in YunoHost.

2.1 Getting access to LDAP in a command

To get access to LDAP you need to authenticate against it, for that you need to declare your command with requiring authentication in the *Principle* this way:

```
configuration:
    authenticate: all
```

Here is a complete example:

```
somecommand:
    category_help: ..
    actions:

        ### somecommand_stuff()
        stuff:
            action_help: ...
            api: GET /...
            configuration:
                authenticate: all
```

This will prompt the user for a password in CLI.

If you only need to **read** LDAP (and not modify it, for example by listing domains), then you prevent the need for a password by using the `ldap-anonymous` authenticator this way:

```
configuration:
    authenticate: all
    authenticator: ldap-anonymous
```

Once you have declared your command like that, your python function will received the `auth` object as first argument, it will be used to talk to LDAP, so you need to declare your function this way:

```
def somecommand_stuff(auth, ...):
    ...
```

2.1.1 auth in the moulinette code

The `auth` object is an instance of `moulinette.authenticators.ldap.Authenticator` class.

Here its docstring:

class moulinette.authenticators.ldap.**Authenticator**(*name*, *vendor*, *parameters*, *extra*)

LDAP Authenticator

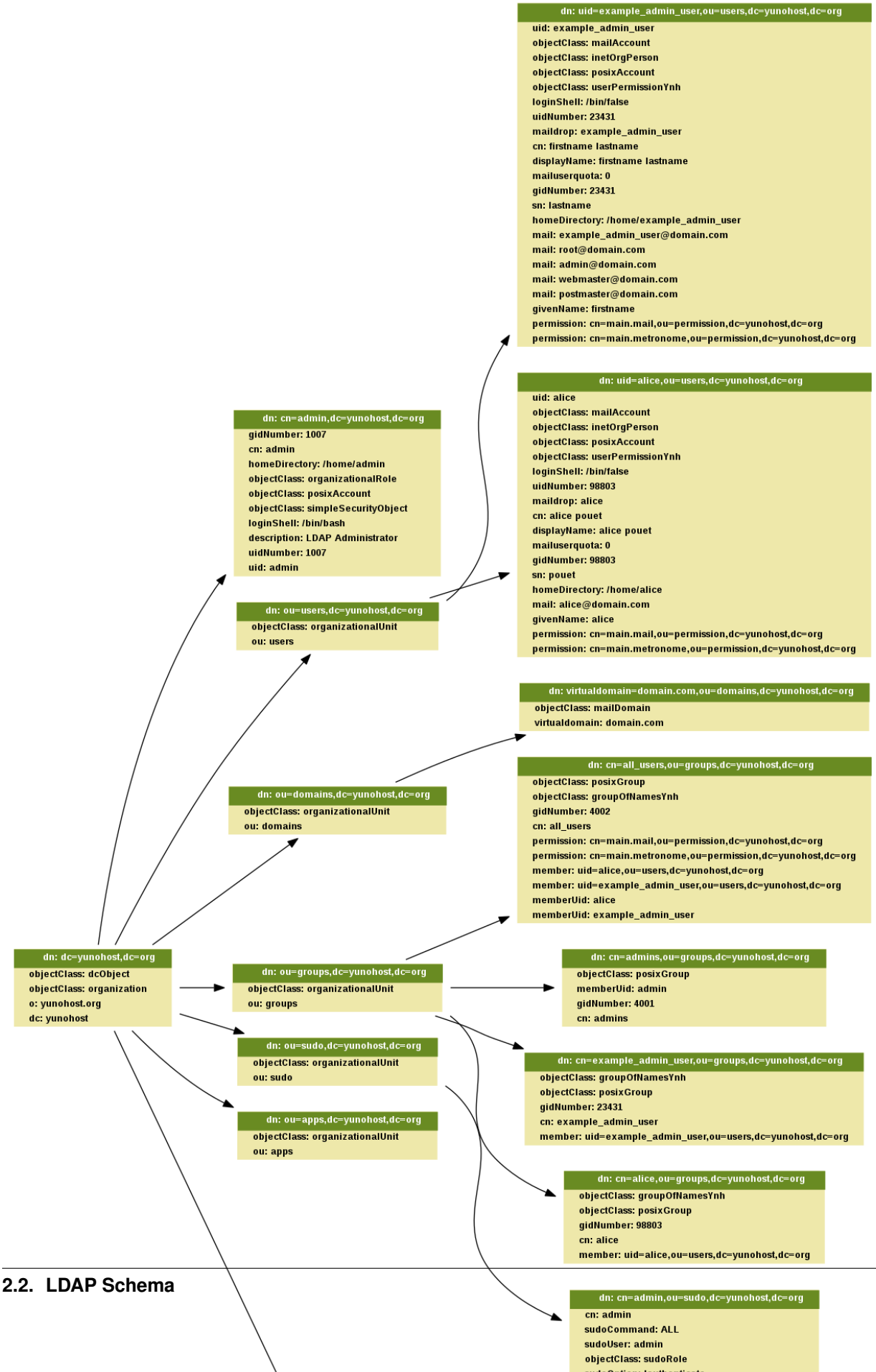
Initialize a LDAP connexion for the given arguments. It attempts to authenticate a user if 'user_rdn' is given - by associating user_rdn and base_dn - and provides extra methods to manage opened connexion.

Keyword arguments:

- uri – The LDAP server URI
- base_dn – The base dn
- user_rdn – The user rdn to authenticate

2.2 LDAP Schema

This is a generated example of the ldap schema provided by YunoHost (to generate this graph uses `make ldap_graph`, you'll need graphviz):


dn: uid=example_admin_user,ou=users,dc=yunohost,dc=org
uid: example_admin_user
objectClass: mailAccount
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: userPermissionYnh
loginShell: /bin/false
uidNumber: 23431
maildrop: example_admin_user
cn: firstname lastname
displayName: firstname lastname
mailuserquota: 0
gidNumber: 23431
sn: lastname
homeDirectory: /home/example_admin_user
mail: example_admin_user@domain.com
mail: root@domain.com
mail: admin@domain.com
mail: webmaster@domain.com
mail: postmaster@domain.com
givenName: firstname
permission: cn=main.mail,ou=permission,dc=yunohost,dc=org
permission: cn=main.metronome,ou=permission,dc=yunohost,dc=org
dn: uid=alice,ou=users,dc=yunohost,dc=org
uid: alice
objectClass: mailAccount
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: userPermissionYnh
loginShell: /bin/false
uidNumber: 98803
maildrop: alice
cn: alice pouet
displayName: alice pouet
mailuserquota: 0
gidNumber: 98803
sn: pouet
homeDirectory: /home/alice
mail: alice@domain.com
givenName: alice
permission: cn=main.mail,ou=permission,dc=yunohost,dc=org
permission: cn=main.metronome,ou=permission,dc=yunohost,dc=org
dn: cn=admin,dc=yunohost,dc=org
gidNumber: 1007
cn: admin
homeDirectory: /home/admin
objectClass: organizationalRole
objectClass: posixAccount
objectClass: simpleSecurityObject
loginShell: /bin/bash
description: LDAP Administrator
uidNumber: 1007
uid: admin
dn: ou=users,dc=yunohost,dc=org
objectClass: organizationalUnit
ou: users
dn: virtualdomain=domain.com,ou=domains,dc=yunohost,dc=org
objectClass: mailDomain
virtualdomain: domain.com
dn: cn=all_users,ou=groups,dc=yunohost,dc=org
objectClass: posixGroup
objectClass: groupOfNamesYnh
gidNumber: 4002
cn: all_users
permission: cn=main.mail,ou=permission,dc=yunohost,dc=org
permission: cn=main.metronome,ou=permission,dc=yunohost,dc=org
member: uid=alice,ou=users,dc=yunohost,dc=org
member: uid=example_admin_user,ou=users,dc=yunohost,dc=org
memberUid: alice
memberUid: example_admin_user
dn: ou=domains,dc=yunohost,dc=org
objectClass: organizationalUnit
ou: domains
dn: dc=yunohost,dc=org
objectClass: dcObject
objectClass: organization
o: yunohost.org
dc: yunohost
dn: ou=groups,dc=yunohost,dc=org
objectClass: organizationalUnit
ou: groups
dn: cn=admins,ou=groups,dc=yunohost,dc=org
objectClass: posixGroup
memberUid: admin
gidNumber: 4001
cn: admins
dn: ou=sudo,dc=yunohost,dc=org
objectClass: organizationalUnit
ou: sudo
dn: cn=example_admin_user,ou=groups,dc=yunohost,dc=org
objectClass: groupOfNamesYnh
objectClass: posixGroup
gidNumber: 23431
cn: example_admin_user
member: uid=example_admin_user,ou=users,dc=yunohost,dc=org
dn: ou=apps,dc=yunohost,dc=org
objectClass: organizationalUnit
ou: apps
dn: cn=alice,ou=groups,dc=yunohost,dc=org
objectClass: groupOfNamesYnh
objectClass: posixGroup
gidNumber: 98803
cn: alice
member: uid=alice,ou=users,dc=yunohost,dc=org
dn: cn=admin,ou=sudo,dc=yunohost,dc=org
cn: admin
sudoCommand: ALL
sudoUser: admin
objectClass: sudoRole

2.3 Reading from LDAP

Reading data from LDAP is done using the `auth` object received as first argument of the python function. To see how to get this object read the previous section.

The API looks like this:

```
auth.search(ldap_path, ldap_query)
```

This will return a list of dictionary with strings as keys and list as values.

You can also specify a list of attributes you want to access from LDAP using a list of string (on only one string apparently):

```
auth.search(ldap_path, ldap_query, ['first_attribute', 'another_attribute'])
```

For example, if we request the user `alice` with its `homeDirectory`, this would look like this:

```
auth.search('ou=users,dc=yunohost,dc=org', '(&(objectclass=person)(uid=alice))', [
↪'homeDirectory', 'another_attribute'])
```

And as a result we will get:

```
[{'homeDirectory': ['/home/alice']}]
```

Notice that even for a single result we get a **list** of result and that every value in the dictionary is also a **list** of values. This is not really convenient and it would be better to have a real ORM, but for now we are stuck with that.

Apparently if we don't specify the list of attributes it seems that we get all attributes (need to be confirmed).

Here is the method docstring:

`Authenticator.`**`search`**(*base=None, filter='(objectClass=*)', attrs=['dn']*)

Search in LDAP base

Perform an LDAP search operation with given arguments and return results as a list.

Keyword arguments:

- base – The dn to search into
- filter – A string representation of the filter to apply
- attrs – A list of attributes to fetch

Returns: A list of all results

2.3.1 Users LDAP schema

According to `ldapvi` this is the user schema (on YunoHost >3.7):

```
# path: uid=the_unix_username,ou=users,dc=yunohost,dc=org
uid: the_unix_username
objectClass: mailAccount
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: userPermissionYnh
loginShell: /bin/false
uidNumber: 80833
```

(continues on next page)

(continued from previous page)

```
maildrop: the_unix_username  # why?
cn: first_name last_name
displayName: first_name last_name
mailuserquota: some_value
gidNumber: 80833
sn: last_name
homeDirectory: /home/the_unix_username
mail: the_unix_username@domain.com
# if the user is the admin he will also have the following mails
mail: root@domain.com
mail: admin@domain.com
mail: webmaster@domain.com
mail: postmaster@domain.com
givenName: first_name
memberOf: cn=the_unix_username,ou=groups,dc=yunohost,dc=org
memberOf: cn=all_users,ou=groups,dc=yunohost,dc=org
permission: cn=main.mail,ou=permission,dc=yunohost,dc=org
permission: cn=main.metronome,ou=permission,dc=yunohost,dc=org
```

The admin user is a special case that looks like this:

```
# path: cn=admin,dc=yunohost,dc=org
gidNumber: 1007
cn: admin
homeDirectory: /home/admin
objectClass: organizationalRole
objectClass: posixAccount
objectClass: simpleSecurityObject
loginShell: /bin/bash
description: LDAP Administrator
uidNumber: 1007
uid: admin
```

Other user related schemas:

```
# path: cn=admins,ou=groups,dc=yunohost,dc=org
objectClass: posixGroup
objectClass: top
memberUid: admin
gidNumber: 4001
cn: admins

# path: cn=admin,ou=sudo,dc=yunohost,dc=org
# this entry seems to specify which unix user is a sudoer
cn: admin
sudoCommand: ALL
sudoUser: admin
objectClass: sudoRole
objectClass: top
sudoOption: !authenticate
sudoHost: ALL
```

2.3.2 Reading users from LDAP

The user schema is located at this path: `ou=users,dc=yunohost,dc=org`

According to already existing code, the queries we uses are:

- `'(&(objectclass=person)(!(uid=root))(!(uid=nobody)))'` to get all users (not that I've never encountered users with `root` or `nobody` uid in the ldap database, those might be there for historical reason)
- `'(&(objectclass=person)(uid=%s))' % username` to access one user data

This give us the 2 following python calls:

```
# all users
auth.search('ou=users,dc=yunohost,dc=org', '(&(objectclass=person)(!(uid=root))(!
↪(uid=nobody)))')

# one user
auth.search('ou=users,dc=yunohost,dc=org', '(&(objectclass=person)(uid=some_username))
↪')
```

Apparently we could also access one user using the following path (and not query): `uid=user_username, ou=users,dc=yunohost,dc=org` but I haven't test it.

If you want specific attributes look at the general documentation on how to read from LDAP a bit above of this section.

2.3.3 Group LDAP schema

According to `ldapvi` this is the user schema (on YunoHost >3.4):

The groups will look like this:

```
dn: cn=the_unix_username,ou=groups,dc=yunohost,dc=org
objectClass: top
objectClass: groupOfNamesYnh
objectClass: posixGroup
gidNumber: 48335
cn: the_unix_username
structuralObjectClass: posixGroup
member: uid=the_unix_username,ou=users,dc=yunohost,dc=org
```

By default you will find in all case a group named *all_users* which will contains all Yunohost users.

```
# path dn: cn=all_users,ou=groups,dc=yunohost,dc=org
objectClass: posixGroup
objectClass: groupOfNamesYnh
gidNumber: 4002
cn: all_users
structuralObjectClass: posixGroup
permission: cn=main.mail,ou=permission,dc=yunohost,dc=org
permission: cn=main.metronome,ou=permission,dc=yunohost,dc=org
member: uid=the_unix_username,ou=users,dc=yunohost,dc=org
memberUid: the_unix_username
```

2.3.4 Reading group from LDAP

The group schema is located at this path: `ou=groups,dc=yunohost,dc=org`

The queries we uses are the 2 following python calls:

```
# all groups
auth.search('ou=groups,dc=yunohost,dc=org', '(objectclass=groupOfNamesYnh)')

# one groups
auth.search(base='ou=groups,dc=yunohost,dc=org', filter='cn=' + groupname)
```

2.3.5 Permission LDAP schema

According to `ldapvi` this is the user schema (on YunoHost >3.4):

The permission will look like this:

```
dn: cn=main.mail,ou=permission,dc=yunohost,dc=org
objectClass: posixGroup
objectClass: permissionYnh
gidNumber: 5001
groupPermission: cn=all_users,ou=groups,dc=yunohost,dc=org
cn: main.mail
structuralObjectClass: posixGroup
memberUid: the_unix_username
inheritPermission: uid=the_unix_username,ou=users,dc=yunohost,dc=org
```

By default you will have a permission for the mail and for metronome. When you install an application a permission also created.

2.3.6 Reading permissions from LDAP

The permission schema is located at this path: `ou=permission,dc=yunohost,dc=org`

The queries we uses are the 2 following python calls:

```
# For all permission
auth.search('ou=permission,dc=yunohost,dc=org', '(objectclass=permissionYnh)')

# For one permission
auth.search(base='ou=permission,dc=yunohost,dc=org', filter='cn=' + permission_name)
```

2.3.7 Domain LDAP schema

According to `ldapvi` this is the domain schema (on YunoHost 2.7):

```
10 virtualdomain=domain.com,ou=domains,dc=yunohost,dc=org
objectClass: mailDomain
objectClass: top
virtualdomain: domain.com
```

2.4 Adding data in LDAP

If you add an object linked to user, group or permission you need run the function *permission_sync_to_user* to keep integrity of permission in LDAP.

Adding stuff in LDAP seems pretty simple, according to existing code it looks like this:

```
auth.add('key=%s,ou=some_location', {'attribute1': 'value', ...})
```

They weird stuff is the path you need to create. This looks like that for domain and users:

```
# domain
auth.add('virtualdomain=%s,ou=domains' % domain, attr_dict)

# user
auth.add('uid=%s,ou=users' % username, attr_dict)
```

You need to respect the expected attributes. Refer to the schema for that.

`auth.add` seems to return something false when it failed (None probably) so you need to check it's return code.

Here is the docstring:

`Authenticator.`**`add`**(*rdn*, *attr_dict*)

Add LDAP entry

Keyword arguments: rdn – DN without domain attr_dict – Dictionnary of attributes/values to add

Returns: Boolean | MoulinetteError

2.4.1 Adding user in LDAP

Here is how it's done for a new user:

```
auth.add('uid=%s,ou=users' % username, {
    'objectClass': ['mailAccount', 'inetOrgPerson', 'posixAccount'],
    'givenName': firstname,
    'sn': lastname,
    'displayName': '%s %s' % (firstname, lastname),
    'cn': fullname,
    'uid': username,
    'mail': mail,
    'maildrop': username,
    'mailuserquota': mailbox_quota,
    'userPassword': user_pwd,
    'gidNumber': uid,
    'uidNumber': uid,
    'homeDirectory': '/home/' + username,
    'loginShell': '/bin/false'
})
```

2.4.2 Adding a domain in LDAP

Here is how it's done for a new domain:

```
auth.add('virtualdomain=%s,ou=domains' % domain, {
    'objectClass': ['mailDomain', 'top']
    'virtualdomain': domain,
})
```

2.5 Updating LDAP data

If you add an object linked to user, group or permission you need run the function *permission_sync_to_user* to keep integrity of permission in LDAP.

Update a user from LDAP looks like a simplified version of searching. The syntax is the following one:

```
auth.update(exact_path_to_object, {'attribute_to_modify': 'new_value', 'another_
↪attribute_to_modify': 'another_value', ...})
```

For example this will update a user `loginShell`:

```
auth.update('uid=some_username,ou=users', {'loginShell': '/bin/bash'})
```

I don't know how this call behave if it fails and what it returns.

Here is the method docstring:

`Authenticator.`**`update`**(*rdn*, *attr_dict*, *new_rdn=False*)

Modify LDAP entry

Keyword arguments: rdn – DN without domain attr_dict – Dictionnary of attributes/values to add new_rdn – New RDN for modification

Returns: Boolean | MoulinetteError

2.5.1 Updating a user in LDAP

This is done this way:

```
auth.update('uid=some_username,ou=users', {'attribute': 'new_value', ...})
```

Refer to the user schema to know which attributes you can modify.

2.6 Validate uniqueness

There is a method to validate the uniqueness of some entry that is used during user creation. It's useful by example to be sure that we have no conflict about email between each user.

Here is how it's used (I don't understand why a path is not provided):

```
# Validate uniqueness of username and mail in LDAP
auth.validate_uniqueness({
    'uid': username,
    'mail': mail
})
```

And here is its docstring:

`Authenticator.`**`update`**(*rdn*, *attr_dict*, *new_rdn=False*)

Modify LDAP entry

Keyword arguments: rdn – DN without domain attr_dict – Dictionnary of attributes/values to add new_rdn – New RDN for modification

Returns: Boolean | MoulinetteError

2.7 Get conflict

Like the last function *validate_uniqueness* but give instead of rising an error this function return which attribute with witch value generate a conflict.

```
# Validate uniqueness of groupname in LDAP
conflict = auth.get_conflict({
    'cn': groupname
}, base_dn='ou=groups,dc=yunohost,dc=org')
if conflict:
    raise YunohostError('group_name_already_exist', name=groupname)
```

2.8 Remove entries from LDAP

If you add an object linked to user, group or permission you need run the function *permission_sync_to_user* to keep integrity of permission in LDAP.

Remove entries from LDAP is very simple, quite close to adding stuff except you don't need to specify the attributes dict, you just need to entrie path:

```
auth.remove(path)
```

Here how it looks like for domain and user:

```
# domain
auth.remove('virtualdomain=%s,ou=domains' % domain)

# user
auth.remove('uid=%s,ou=users' % username)
```

`auth.remove` returns something that evaluate to False when it fails (`None` ?) so you need to check it returns code.

`Authenticator.`**`remove`**`(`*`rdn`*`)`

Remove LDAP entry

Keyword arguments: rdn – DN without domain

Returns: Boolean | MoulinetteError

2.9 Reading LDIF file

Reading parsing a ldif to be able to insert in the LDAP database is really easy. Here is how to get the content of a LDIF file

```
from moulinette.utils.filesystem import read_ldif

my_reslut = read_ldif("your_file.ldif")
```

Note that the main difference of what the auth object return with the search method is that this function return a 2-tuples with the "dn" and the LDAP entry.

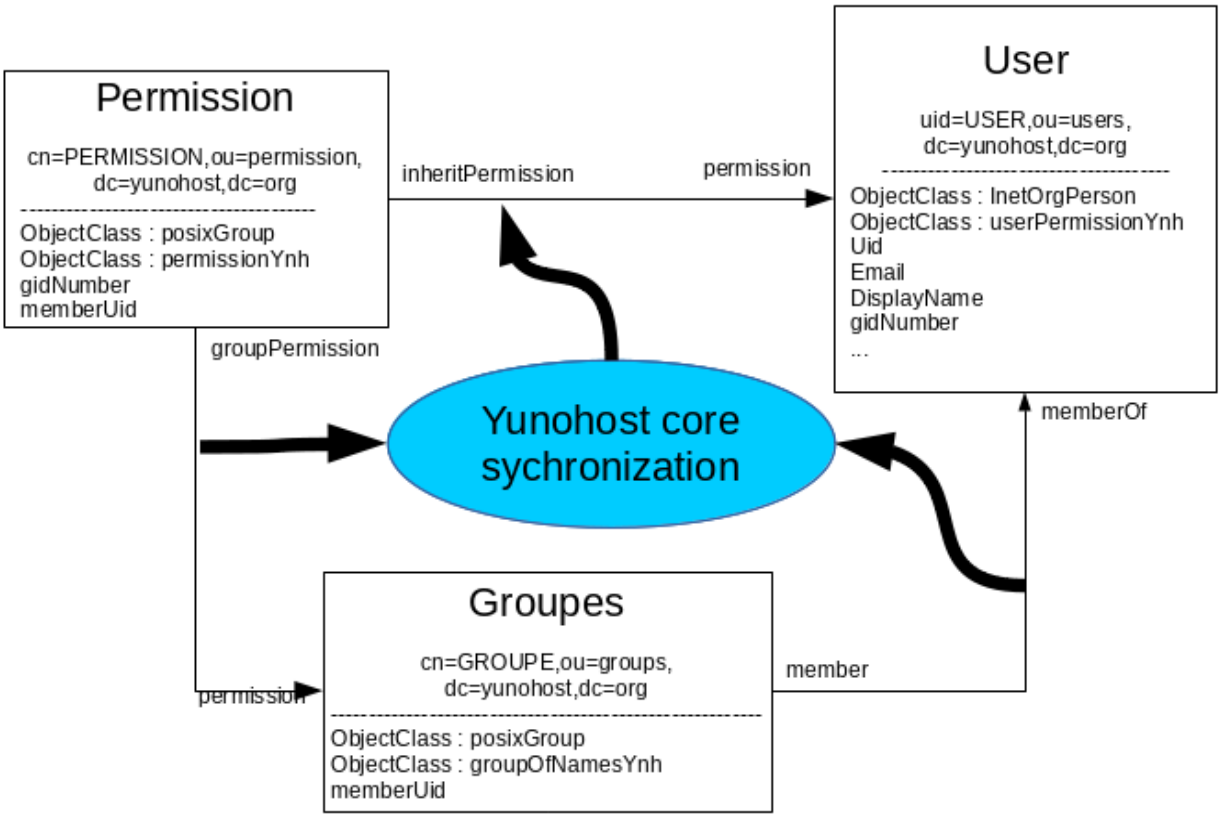
CHAPTER 3

LDAP architecture in Yunohost

In Yunohost to be able to manage the user and the permission we use 3 parts:

- User object
- Permission object
- Group object

We can see the interaction between these object as this following:



As you can see there are link between these 3 objets:

- The first link is between the user and the group. It define which user is in which group. Note that all user has a group with his name. Note that in all Yunohost instance you have a group named *all_users*. In this group you will find all Yunohost users.
- The second link is between the permission and the groups. This link is defined by the administrator. By default all permission are linked to the group *all_users*, so all user will be allowed to access to this permission.
- The third link between the User and the Permission is more technical. It give the possibility to the application to get a list of all user allowed to access to. This link is dynamically generated by core. The function *permission_sync_to_user* in the module *permission* do this work.

The option *force* of the function *permission_sync_to_user* is used when you add the data to LDAP with *slapadd*. *slapadd* update the LDAP database without the LDAP demon process. The advantage of this is that you can bypass the integrity check (like the link between the object by the memberOf overlay). The disadvantage is that the the memberOf overlay wont update anything so if you don't fix the integrity after after to run *slapadd*, the permission in LDAP might be corrupted. Running the function permission_sync_to_user' with the option *force* will do this work to fix all integrity error.

To be able to have an attribute in both is of theses 3 link we use the *memberOf* overlay in LDAP. This following line define the configuration to have these 3 link dynamically updated :

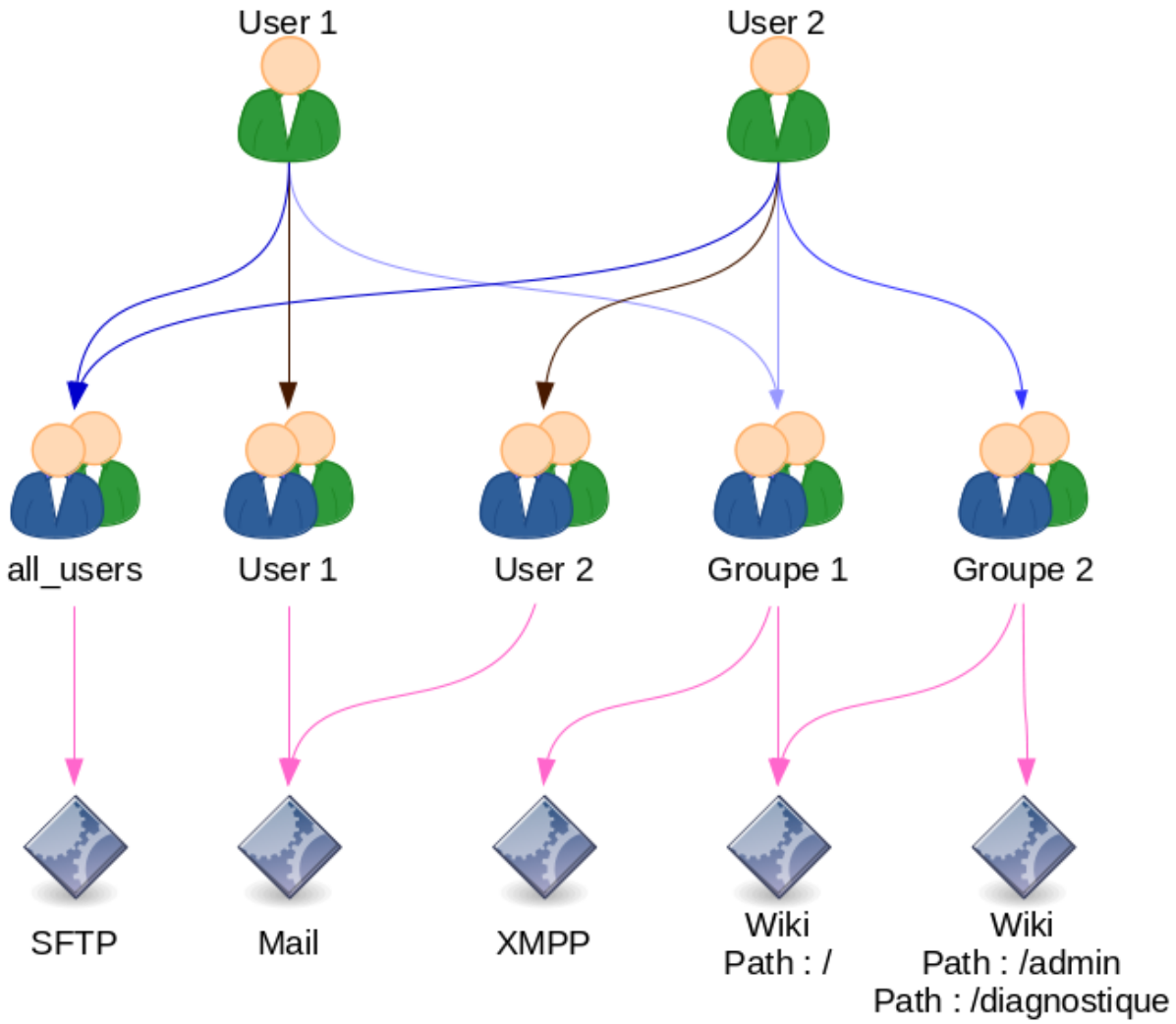
```
# Link user <-> group
#dn: olcOverlay={0}memberof,olcDatabase={1}mdb,cn=config
overlay                 memberof
memberof-group-oc       groupOfNamesYnh
memberof-member-ad      member
memberof-memberof-ad    memberOf
memberof-dangling       error
```

(continues on next page)

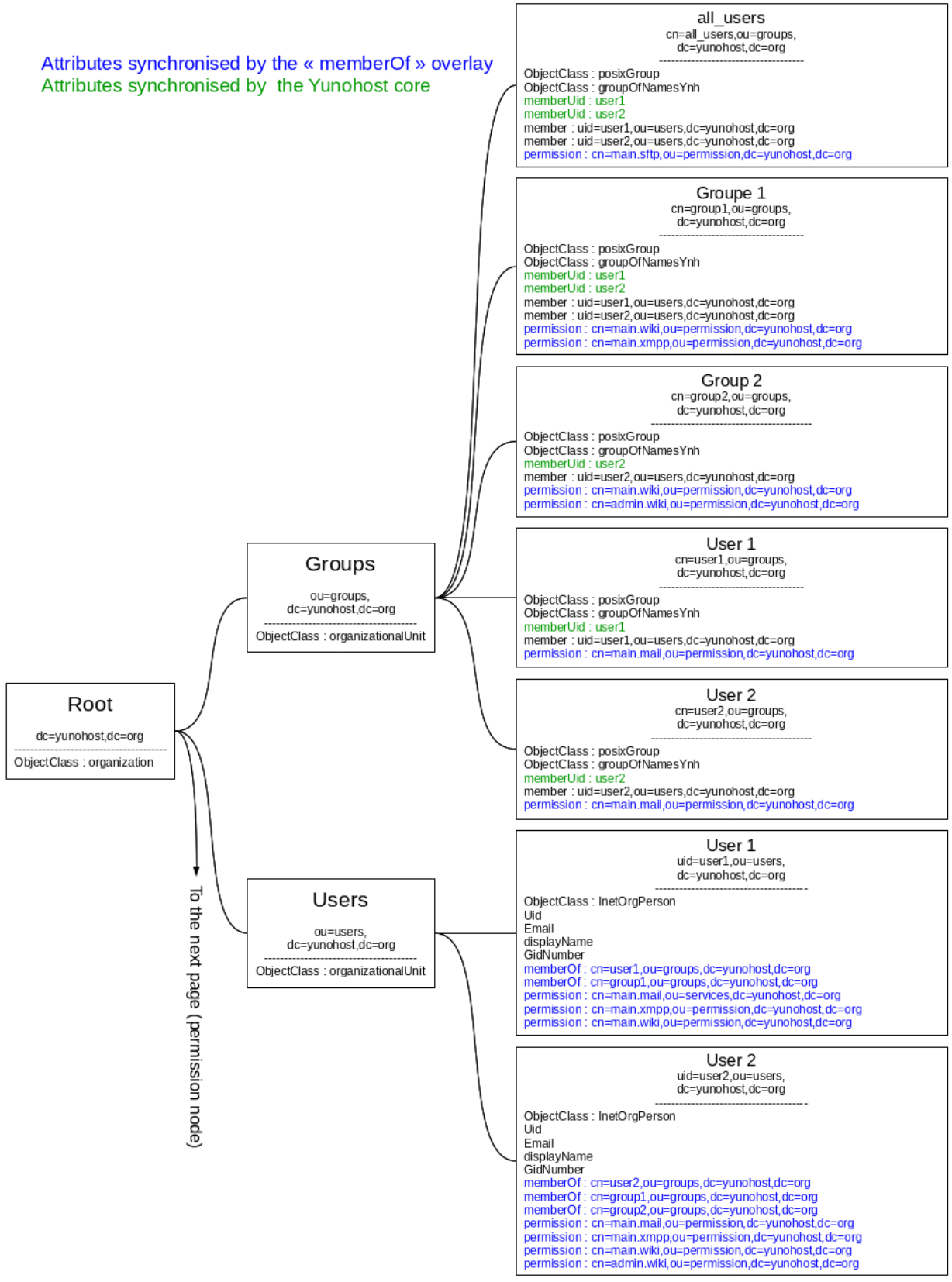
(continued from previous page)

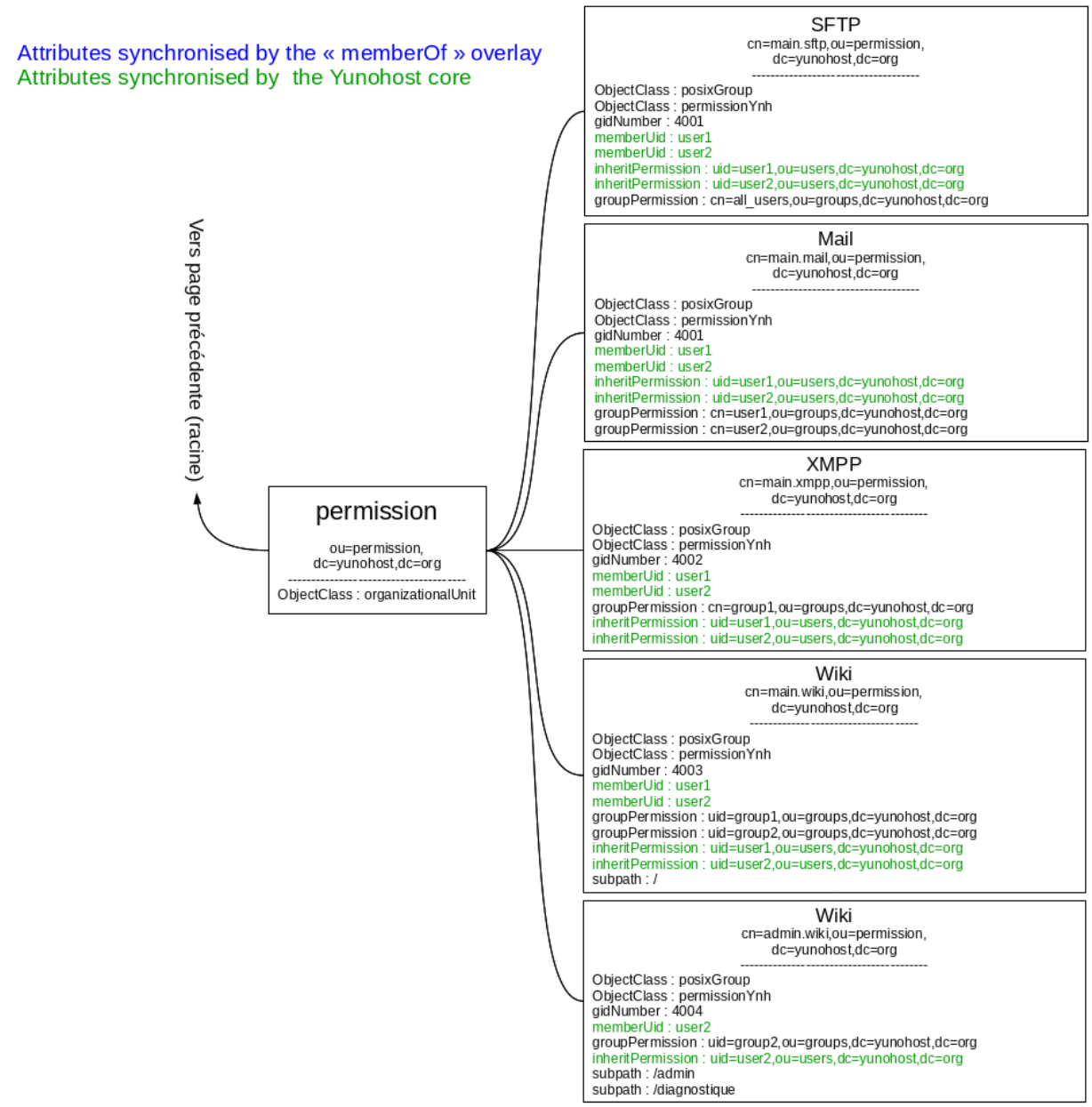
```
memberof-refint         TRUE

# Link permission <-> groupes
#dn: olcOverlay={1}memberof,olcDatabase={1}mdb,cn=config
overlay                 memberof
memberof-group-oc       permissionYnh
memberof-member-ad      groupPermission
memberof-memberof-ad    permission
memberof-dangling       error
memberof-refint         TRUE

# Link permission <-> user
#dn: olcOverlay={2}memberof,olcDatabase={1}mdb,cn=config
overlay                 memberof
memberof-group-oc       permissionYnh
memberof-member-ad      inheritPermission
memberof-memberof-ad    permission
memberof-dangling       error
memberof-refint         TRUE
```

This foolwing example show how will be represented in LDAP as simple concept of permission.



This schema show what will be in LDAP in these following schema:


Attributes synchronised by the « memberOf » overlay
Attributes synchronised by the Yunohost core
Root
dc=yunohost,dc=org
ObjectClass : organization
Groups
ou=groups,
dc=yunohost,dc=org
ObjectClass : organizationalUnit
Users
ou=users,
dc=yunohost,dc=org
ObjectClass : organizationalUnit
To the next page (permission node)
all_users
cn=all_users,ou=groups,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : groupOfNamesYnh
memberUid : user1
memberUid : user2
member : uid=user1,ou=users,dc=yunohost,dc=org
member : uid=user2,ou=users,dc=yunohost,dc=org
permission : cn=main.sftp,ou=permission,dc=yunohost,dc=org
Groupe 1
cn=group1,ou=groups,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : groupOfNamesYnh
memberUid : user1
memberUid : user2
member : uid=user1,ou=users,dc=yunohost,dc=org
member : uid=user2,ou=users,dc=yunohost,dc=org
permission : cn=main.wiki,ou=permission,dc=yunohost,dc=org
permission : cn=main.xmpp,ou=permission,dc=yunohost,dc=org
Group 2
cn=group2,ou=groups,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : groupOfNamesYnh
memberUid : user2
member : uid=user2,ou=users,dc=yunohost,dc=org
permission : cn=main.wiki,ou=permission,dc=yunohost,dc=org
permission : cn=admin.wiki,ou=permission,dc=yunohost,dc=org
User 1
cn=user1,ou=groups,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : groupOfNamesYnh
memberUid : user1
member : uid=user1,ou=users,dc=yunohost,dc=org
permission : cn=main.mail,ou=permission,dc=yunohost,dc=org
User 2
cn=user2,ou=groups,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : groupOfNamesYnh
memberUid : user2
member : uid=user2,ou=users,dc=yunohost,dc=org
permission : cn=main.mail,ou=permission,dc=yunohost,dc=org
User 1
uid=user1,ou=users,
dc=yunohost,dc=org
ObjectClass : InetOrgPerson
Uid
Email
displayName
GidNumber
memberOf : cn=user1,ou=groups,dc=yunohost,dc=org
memberOf : cn=group1,ou=groups,dc=yunohost,dc=org
permission : cn=main.mail,ou=services,dc=yunohost,dc=org
permission : cn=main.xmpp,ou=permission,dc=yunohost,dc=org
permission : cn=main.wiki,ou=permission,dc=yunohost,dc=org
User 2
uid=user2,ou=users,
dc=yunohost,dc=org
ObjectClass : InetOrgPerson
Uid
Email
displayName
GidNumber
memberOf : cn=user2,ou=groups,dc=yunohost,dc=org
memberOf : cn=group1,ou=groups,dc=yunohost,dc=org
memberOf : cn=group2,ou=groups,dc=yunohost,dc=org
permission : cn=main.mail,ou=permission,dc=yunohost,dc=org
permission : cn=main.xmpp,ou=permission,dc=yunohost,dc=org
permission : cn=main.wiki,ou=permission,dc=yunohost,dc=org
permission : cn=admin.wiki,ou=permission,dc=yunohost,dc=org


Attributes synchronised by the « memberOf » overlay
Attributes synchronised by the Yunohost core
Vers page précédente (racine)
permission
ou=permission,
dc=yunohost,dc=org
ObjectClass : organizationalUnit
SFTP
cn=main.sftp,ou=permission,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : permissionYnh
gidNumber : 4001
memberUid : user1
memberUid : user2
inheritPermission : uid=user1,ou=users,dc=yunohost,dc=org
inheritPermission : uid=user2,ou=users,dc=yunohost,dc=org
groupPermission : cn=all_users,ou=groups,dc=yunohost,dc=org
Mail
cn=main.mail,ou=permission,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : permissionYnh
gidNumber : 4001
memberUid : user1
memberUid : user2
inheritPermission : uid=user1,ou=users,dc=yunohost,dc=org
inheritPermission : uid=user2,ou=users,dc=yunohost,dc=org
groupPermission : cn=user1,ou=groups,dc=yunohost,dc=org
groupPermission : cn=user2,ou=groups,dc=yunohost,dc=org
XMPP
cn=main.xmpp,ou=permission,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : permissionYnh
gidNumber : 4002
memberUid : user1
memberUid : user2
groupPermission : cn=group1,ou=groups,dc=yunohost,dc=org
inheritPermission : uid=user1,ou=users,dc=yunohost,dc=org
inheritPermission : uid=user2,ou=users,dc=yunohost,dc=org
Wiki
cn=main.wiki,ou=permission,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : permissionYnh
gidNumber : 4003
memberUid : user1
memberUid : user2
groupPermission : uid=group1,ou=groups,dc=yunohost,dc=org
groupPermission : uid=group2,ou=groups,dc=yunohost,dc=org
inheritPermission : uid=user1,ou=users,dc=yunohost,dc=org
inheritPermission : uid=user2,ou=users,dc=yunohost,dc=org
subpath : /
Wiki
cn=admin.wiki,ou=permission,
dc=yunohost,dc=org
ObjectClass : posixGroup
ObjectClass : permissionYnh
gidNumber : 4004
memberUid : user2
groupPermission : uid=group2,ou=groups,dc=yunohost,dc=org
inheritPermission : uid=user2,ou=users,dc=yunohost,dc=org
subpath : /admin
subpath : /diagnostique

CHAPTER 4

LDAP integration in Yunohost applications

To have a complete integration of LDAP in your application you need to configure LDAP as follow :

```
Host:                ldap://localhost
Port:                389
Base DN:             dc=yunohost,dc=org
User DN:             ou=users,dc=yunohost,dc=org
Group DN:            ou=groups,dc=yunohost,dc=org
fiter :              (&(objectClass=posixAccount)(permission=cn=YOUR_APP.main,
↪ou=permission,dc=yunohost,dc=org))
LDAP Username:       uid
LDAP Email Address: mail
```

By this your application will get the list of all user allowed to access to your application.

CHAPTER 5

Translations using the m18n object

The moulinette provides a way to do translations and YunoHost uses it. This is done via the *m18n* object that you can import this way:

```
from moulinette import m18n
```

The *m18n* object comes with 2 method:

- *m18n.n* to uses for translations within YunoHost
- *m18n.g* to uses for translations within Moulinette itself

Their API is identical.

Here are example of uses:

```
m18n.n('some_translation_key')
m18n.g('some_translation_key')

m18n.n('some_translation_key', string_formating_argument_1=some_variable)
m18n.g('some_translation_key', string_formating_argument_1=some_variable)
```

The translation key must be present in *locales/en.json* of either YunoHost (for *.n*) or moulinette (for *.g*).

5.1 Docstring

As a reference, here are the docstrings of the m18n class:

class moulinette.core.**Moulinette18n**(*default_locale='en'*)

Internationalization service for the moulinette

Manage internationalization and access to the proper keys translation used in the moulinette and libraries.

Keyword arguments:

- package – The current Package instance

- default_locale – The default locale to use

Moulinette18n.**n**(*key*, **args*, ***kwargs*)
Retrieve proper translation for a moulinette key

Attempt to retrieve value for a key from current loaded namespace translations using the current locale or the default one if 'key' is not found.

Keyword arguments:

- key – The key to translate

Moulinette18n.**g**(*key*, **args*, ***kwargs*)
Retrieve proper translation for a moulinette key

Attempt to retrieve value for a key from moulinette translations using the current locale or the default locale if 'key' is not found.

Keyword arguments:

- key – The key to translate

CHAPTER 6

File system operation utils

moulinette.utils.filesystem.**read_file**(*file_path*)
Read a regular text file

Keyword argument: file_path – Path to the text file

moulinette.utils.filesystem.**read_json**(*file_path*)
Read a json file

Keyword argument: file_path – Path to the json file

moulinette.utils.filesystem.**read_yaml**(*file_path*)
Safely read a yaml file

Keyword argument: file_path – Path to the yaml file

moulinette.utils.filesystem.**read_toml**(*file_path*)
Safely read a toml file

Keyword argument: file_path – Path to the toml file

moulinette.utils.filesystem.**write_to_file**(*file_path*, *data*, *file_mode='w'*)
Write a single string or a list of string to a text file. The text file will be overwritten by default.

Keyword argument: file_path – Path to the output file data – The data to write (must be a string or list of string) file_mode – Mode used when writing the file. Option meant to be used by append_to_file to avoid duplicating the code of this function.

moulinette.utils.filesystem.**append_to_file**(*file_path*, *data*)
Append a single string or a list of string to a text file.

Keyword argument: file_path – Path to the output file data – The data to write (must be a string or list of string)

moulinette.utils.filesystem.**write_to_json**(*file_path*, *data*)
Write a dictionnary or a list to a json file

Keyword argument: file_path – Path to the output json file data – The data to write (must be a dict or a list)

moulinette.utils.filesystem.**mkdir**(*path*, *mode=511*, *parents=False*, *uid=None*, *gid=None*, *force=False*)

Create a directory with optional features

Create a directory and optionaly set its permissions to mode and its owner and/or group. If path refers to an existing path, nothing is done unless force is True.

Keyword arguments:

- path – The directory to create
- mode – Numeric path mode to set
- parents – Make parent directories as needed
- uid – Numeric uid or user name
- gid – Numeric gid or group name
- force – Force directory creation and owning even if the path exists

moulinette.utils.filesystem.**chown**(*path*, *uid=None*, *gid=None*, *recursive=False*)

Change the owner and/or group of a path

Keyword arguments:

- uid – Numeric uid or user name
- gid – Numeric gid or group name
- recursive – Operate on path recursively

moulinette.utils.filesystem.**chmod**(*path*, *mode*, *fmode=None*, *recursive=False*)

Change the mode of a path

Keyword arguments:

- mode – Numeric path mode to set
- fmode – Numeric file mode to set in case of a recursive directory
- recursive – Operate on path recursively

moulinette.utils.filesystem.**rm**(*path*, *recursive=False*, *force=False*)

Remove a file or directory

Keyword arguments:

- path – The path to remove
- recursive – Remove directories and their contents recursively
- force – Ignore nonexistent files

CHAPTER 7

Network operation utils

moulinette.utils.network.**download_text**(*url*, *timeout=30*, *expected_status_code=200*)
Download text from a url and returns the raw text

Keyword argument: url – The url to download the data from timeout – Number of seconds allowed for download to effectively start before giving up expected_status_code – Status code expected from the request. Can be None to ignore the status code.

moulinette.utils.network.**download_json**(*url*, *timeout=30*, *expected_status_code=200*)
Download json from a url and returns the loaded json object

Keyword argument: url – The url to download the data from timeout – Number of seconds allowed for download to effectively start before giving up

CHAPTER 8

Process operation utils

moulinette.utils.process.**check_output**(*args*, *stderr=-2*, *shell=True*, ***kwargs*)
Run command with arguments and return its output as a byte string

Overwrite some of the arguments to capture standard error in the result and use shell by default before calling subprocess.check_output.

moulinette.utils.process.**call_async_output**(*args*, *callback*, ***kwargs*)
Run command and provide its output asynchronously

Run command with arguments and wait for it to complete to return the returncode attribute. The *callback* can be a method or a 2-tuple of methods - for stdout and stderr respectively - which must take one byte string argument. It will be called each time the command produces some output.

The stdout and stderr additional arguments for the Popen constructor are not allowed as they are used internally.

Keyword arguments:

- args – String or sequence of program arguments
- callback – Method or object to call with output as argument
- kwargs – Additional arguments for the Popen constructor

Returns: Exit status of the command

moulinette.utils.process.**run_commands**(*cmds*, *callback=None*, *separate_stderr=False*, *shell=True*, ***kwargs*)
Run multiple commands with error management

Run a list of commands and allow to manage how to treat errors either with raise_on_error or callback arguments.

If callback is provided, it will be called when the command returns a non-zero exit code. The callback must take 3 arguments; the returncode, the command which failed and the command output. The callback should return either False to stop commands execution or True to continue. Otherwise, if raise_on_error is True a CalledProcessError exception will be raised when a command returns a non-zero exit code.

If callback is provided or raise_on_error is False, all commands will be executed and the number of failed commands will be returned.

The standard output and error of a failed command can be separated with separate_stderr set to True. In that case, the output argument passed to the callback or the output attribute of the CalledProcessError exception will be a 2-tuple containing stdout and stderr as byte strings.

Keyword arguments:

- cmds – List of commands to run
- **callback – Method or object to call on command failure. If no** callback is given, a "subprocess.CalledProcessError" will be raised in case of command failure.
- separate_stderr – True to return command output as a 2-tuple
- kwargs – Additional arguments for the Popen constructor

Returns: Number of failed commands

CHAPTER 9

Stream operation utils

moulinette.utils.stream.**async_file_reading**(*fd*, *callback*)
Helper which instantiate and run an AsynchronousFileReader.

CHAPTER 10

Text operation utils

moulinette.utils.text.**search**(*pattern*, *text*, *count=0*, *flags=0*)
Search for pattern in a text

Scan through text looking for all locations where the regular expression pattern matches, and return them as a list of strings.

The optional argument count is the maximum number of pattern occurences to return; count must be an integer. If omitted or zero, all occurences will be returned. If it's a negative number, occurences to return will be counted backward. If only one occurence is requested, it will be returned as a string.

The expression's behaviour can be modified by specifying a flags value. Refer to the re module documentation for available variables.

moulinette.utils.text.**searchf**(*pattern*, *path*, *count=0*, *flags=8*)
Search for pattern in a file

Map the file with given path to memory and search for pattern in it content by using the search function.

moulinette.utils.text.**prependlines**(*text*, *prepend*)
Prepend a string to each line of a text

moulinette.utils.text.**random_ascii**(*length=20*)
Return a random ascii string

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index