# PyLDAP ORM Documentation

***Release 0.0.1***

**Bruno Bonfils**

**Oct 15, 2016**

PyLDAP_ORM is a Python3 module, based on pyldap, to help uses of python classes to interact with a LDAP server.

CHAPTER 1

# Getting started

Before using PyORM, there are three steps:

- Define models
- Create a session
- Perform authentication

## 1.1 Models

In a first time, you must define which kind of business objects you want to manage. Some base models are available in the `pyldap_orm.models` module.

```
from pyldap_orm import LDAPSession
from pyldap_orm import models

class LDAPUser(models.LDAPModelUser):
    required_attributes = ['cn', 'uid']
    required_objectclasses = ['inetOrgPerson']
    base = 'ou=People,dc=OpenCSI,dc=com'


class LDAPUsers(models.LDAPModelUsers):
    children = LDAPUser
```

## 1.2 Session

Then, you need to create a connection to the LDAP server, using `LDAPSession` object.

The following connection methods are available:

- LDAP (url start with `ldap://`)
- LDAPs (url start with `ldaps://`)
- STARTTLS (url with `ldap://`, and `mode=LDAPSession.STARTTLS`)

You can also provide `cert` and `key` arguments to provide a client certificate negociation. This is required to perform `AUTH_SASL_EXTERNAL` authentication.

Listing 1.1: Plain LDAP session

```
session = LDAPSession(backend='ldap://localhost:1389/')
```

Listing 1.2: LDAPS session

```
session = LDAPSession(backend='ldaps://localhost:1636/')
```

Listing 1.3: StartTLS session

```
session = LDAPSession(backend='ldap://localhost:1389/', mode=LDAPSession.STARTTLS)
```

Listing 1.4: StartTLS with client certificate

```
session = LDAPSession(backend='ldap://localhost:1389/',
                      mode=LDAPSession.STARTTLS,
                      cert='/home/asyd/Downloads/bbonfils-test.pem',
                      key='/home/asyd/Downloads/bbonfils-test.pem')
```

## 1.3 Authentication

And then, you need to authenticate to the server.

The following authentication methods are available:

- Anonymous binding (no `bind_dn` and `credential` provided)
- Simple bind (`bind_dn` and `credential` must be set)
- SASL EXTERNAL (define `mode=LDAPSession.AUTH_SASL_EXTERNAL`, `cert` and `key` must be provided at the session layer)

Listing 1.5: Simple bind authentication

```
session.authenticate(bind_dn='cn=LDAP Manager,ou=Services,dc=OpenCSI,dc=com',
                     credential='password')
```

Listing 1.6: SASL EXTERNAL authentication

```
session.authenticate(mode=LDAPSession.AUTH_SASL_EXTERNAL)
```

## 1.4 Search

Finally, you can now performs some search. For example by `uid` attribute, and print the user's dn using the following code:

```
user = LDAPUser(session).by_attr('uid', 'asyd')
print(user.dn)
```

## 1.5 Accessing attributes

### 1.5.1 Attributes mapping

| Attribute OID | Attribute desc. | Conversion |
|---|---|---|
| 1.3.6.1.4.1.1466.115.121.1.7 | Boolean | bool |
| 1.3.6.1.4.1.1466.115.121.1.12 | DN | str |
| 1.3.6.1.4.1.1466.115.121.1.15 | Directory String | str |
| 1.3.6.1.4.1.1466.115.121.1.26 | IA String | str |
| 1.3.6.1.4.1.1466.115.121.1.27 | Integer | int |
| 1.3.6.1.4.1.1466.115.121.1.37 | Object Class | str |
| 1.3.6.1.4.1.1466.115.121.1.38 | OID | str |
| 1.3.6.1.4.1.1466.115.121.1.50 | Telephone number | str |

CHAPTER 2

# Full example

```
#!/usr/bin/env python3
# coding: UTF-8

from pyldap_orm import LDAPSession, LDAPObject
from pyldap_orm import models
import pyldap_orm.controls
import logging
import os


class LDAPUser(models.LDAPModelUser):
    base = 'ou=People,dc=example,dc=com'
    required_attributes = ['cn']
    required_objectclasses = ['inetOrgPerson']
    membership_attribute = 'isMemberOf'

    def check(self):
        """
        Remove groups that doesn't belong to ou=Groups,dc=example,dc=com
        """
        try:
            for group in getattr(self, self.membership_attribute):
                if LDAPGroup.base not in group:
                    getattr(self, self.membership_attribute).remove(group)
        except KeyError:
            pass


class LDAPUsers(models.LDAPModelUsers):
    children = LDAPUser


class LDAPGroup(models.LDAPModelGroup):
    base = 'ou=Groups,dc=example,dc=com'
    required_attributes = ['cn']


class LDAPGroups(models.LDAPModelList):
    children = LDAPGroup


def print_entry(entry, extended=False):
```

```
    """
    Print on stdout a LDAP entry.

    :param entry: a LDAPObject instance
    :param extended: if true, also display entry attributes, otherwise only display␣
↪its DN.
    :type entry: LDAPObject
    """
    print(entry.dn)
    if extended:
        for attribute in entry.attributes():
            print(" {}: {}".format(attribute, [value for value in getattr(entry,␣
↪attribute)]))


def main():
    logging.basicConfig(level=logging.INFO)
    # Connect using client certificate, and use a SASL binding, using EXTERNAL␣
↪mechanism.
    # By default, pyldap_orm will used /etc/ssl/certs
    cwd = os.path.dirname(os.path.realpath(__file__))
    session = LDAPSession(backend='ldap://localhost:9389/',
                          mode=LDAPSession.STARTTLS,
                          cacertdir=None,
                          cert='{}/tests/extra/tls/client.pem'.format(cwd),
                          key='{}/tests/extra/tls/client.pem'.format(cwd))

    print("SASL EXTERNAL authentication")
    session.authenticate(mode=LDAPSession.AUTH_SASL_EXTERNAL)
    print("Whoami: {}".format(session.whoami()))

    user = LDAPUser(session).by_attr('uid', 'bbo')
    print_entry(user, extended=True)

    print("\nAll users, sorted by uid:")
    for user in LDAPUsers(session).all(serverctrls=[pyldap_orm.controls.
↪ServerSideSort(['uid'])]):
        print_entry(user)

    print("\nGroups:")
    for group in LDAPGroups(session).all():
        print("{} {}".format(group.cn[0], group.dn))

    print("\nMembers of group dn: cn=Developers,ou=Groups,dc=example,dc=com")
    for user in LDAPUsers(session).by_dn_membership(dn=
↪'cn=Developers,ou=Groups,dc=example,dc=com'):
        print_entry(user)

    print("\nMembers of group name: Developers")
    for user in LDAPUsers(session).by_name_membership('Developers', LDAPGroup):
        print_entry(user)

    print("\n-----")
    # Reconnect using LDAPS and simple bind
    print("Reconnect using LDAPs and simple authentication")
    session = LDAPSession(backend='ldaps://localhost:9636/', cacertdir=None)
    session.authenticate(bind_dn=
↪'cn=ldapmanager,ou=Services,ou=People,dc=example,dc=com',
```

```
                        credential='password')

    print("Whoami: {}".format(session.whoami()))
    print("\nBy DN, display all attributes including operational ones")
    self_entry = LDAPObject(session).by_dn(
↪'cn=ldapmanager,ou=Services,ou=People,dc=example,dc=com', attributes=['*', '+'])
    print_entry(self_entry, extended=True)
    self_entry.description = ['Toto']
    # self_entry.save()

    # Create a new entry
    new_user = LDAPUser(session)
    new_user.dn = 'cn=Vladimir Poutine,ou=Employees,ou=People,dc=example,dc=com'
    new_user.sn = ['Poutine']
    new_user.givenName = ['Validimir']
    new_user.cn = ["{} {}".format(new_user.givenName[0], new_user.sn[0])]
    new_user.uid = ['vpoutine']
    new_user.mail = ['poutine@cei.com']
    new_user.save()
    # But delete it after
    new_user.delete()

if __name__ == '__main__':
    main()
```

CHAPTER 3

# Session

**class** pyldap_orm.session.**LDAPSession**(*backend*, *mode=0*, *cert=None*, *key=None*, *cacertdir='/etc/ssl/certs'*)

Create a LDAPSession by connecting to the LDAP server.

**Tested servers:**

- OpenDJ
- OpenLDAP

A basic usage looks like:

```
>>> session = LDAPSession(backend='ldap://localhost:389', mode=LDAPSession.
↪STARTTLS)
>>> session.authenticate('cn=admin,dc=example,dc=com', 'password')
```

You can also bind as anonymous:

```
>>> session.authenticate()
```

**Parameters**

- **backend** – a LDAP URI like ldaps?://host(:port)?
- **mode** – Transport mode, must be LDAPSession.PLAIN (the default), LDAPSession.STARTTLS or LDAPSession.LDAPS
- **cert** – An optional client certificate, in PEM format
- **key** – The client certificate related private key, in PEM format with no password
- **cacertdir** – Directory of CA certificates, default is /etc/ssl/certs

**authenticate**(*bind_dn=None*, *credential=None*, *mode=0*)

Perform LDAP authentication and parse schema. This method is mandatory.

**Parameters**

- **bind_dn** – optional string to perform a bind
- **credential** – optional string with the password of bind_dn
- **mode** – Can se LDAPSession.AUTH_SIMPLE_BIND (the default) or LDAPSession.AUTH_SASL_EXTERNAL

**parse_schema**()
Create `self.schema['attributes']` dictionary where values are a tuple holding the syntax oid and a boolean (true if the attribute is single valued).

**search**(*base*, *scope=<Mock id='139667285834552'>*, *ldap_filter='(objectClass=*)'*, *attributes=None*, *serverctrls=None*)
Perform a low level LDAP search (synchronous) using the given arguments.

**Parameters**

- **base** – Base DN of the search
- **scope** – Scope of the search, default is SCOPE_SUBTREE
- **ldap_filter** – ldap filter, default is '(objectClass=*)'
- **attributes** – An array of attributes to return, default is ['*']
- **serverctrls** – An array server extended controls

**Returns** a list of tuples (dn, attributes)

CHAPTER 4

# Core

This is core of pyldap_orm.

**class** pyldap_orm.core.**LDAPModelList**(*session=None*)
Manages a list of LDAPObject instances.

**Parameters** **session** (LDAPSession) – An optional instance of LDAPSession.

**by_attr**(*attr*, *value*, *attributes=None*, *serverctrls=None*)
Search an object of class cls by adding a LDAP filter (&(..)(attr=value))

**Parameters**

- **attr** – Attribute to search
- **value** – Attribute value
- **attributes** – An optional array of the expected attributes returned by the search
- **serverctrls** – An optional array with attributes to request server side sorting

**Returns** A list of self.children

**Return type** list

**class** pyldap_orm.core.**LDAPObject**(*session*)
LDAPObject is one of the core class of the ORM. It represent an LDAP object.

**Parameters** **session** (LDAPSession) – an optional LDAPSession instance used to perform operations on a LDAP server.

**attributes**()
Return the list of attributes existing for the current instance

**Returns** List of attributes

**Return type** list

**by_attr**(*attr*, *value*, *attributes=None*)
Search an object by adding a LDAP filter (&(..)(attr=value), where (..) is the search attribute model filter, like '(objectClass=inetOrgPerson)' for a user.

**Parameters**

- **attr** – Attribute to search, like uid, givenName, etc.
- **value** – Attribute value
- **attributes** – Optional array of attributes to returned, if none, all standard attributes are returned.

**Returns** an instance of class cls

**by_dn**(*dn*, *attributes=None*)
Request an object by its DN.

**Parameters**

- **dn** – DN of the LDAP object to query
- **attributes** – Optional array of attributes to returned, if none, all standard attributes are returned.

**Returns** An instance of current LDAPObject inheritance

**check**()
Override this method to perform post operations like remove unwanted values or perform some business checks.

For example, if you want to remove groups that doesn't belong to your LDAPGroup.base you can use the following code:

```
for group in getattr(self, self.membership_attribute):
    if LDAPGroup.base not in group:
    getattr(self, self.membership_attribute).remove(group)
```

**classmethod filter**()
Compute the filter regarding given required_attributes and required_objectclasses class attributes.

**Returns** A string that hold the LDAP filter.

**parse**(*entry*)
This method fill attributes and dn of current instance.

Then, the _check function is called to make some tests, like test if each required_attributes are present.

**Parameters** **entry** – a LDAP entry

**parse_single**(*entries*)
Parse the first entry of entries by calling parse instance method.

**Parameters** **entries**(*list*) –

**Returns**

**save**()
This method is a little magic. Depending on the object state you called it, it can create or update an existing object.

There is even more magic when you create a new object. If the _dn attribute is not set (None), it will be computed from the name_attribute, and the base.

If there is no objectClass defined, the required_objectclasses will be used.

Last, verify that all attributes from required_attributes exists.

CHAPTER 5

# Core models

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

## p

## A



## B



## C



## F



## L



## P



## S